TARCIZIO ALEXANDRE BINI

ANÁLISE DA APLICABILIDADE DAS REGRAS DE OURO AO TUNING DE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

Tese apresentada como requisito parcial à obtenção do título de Doutor em Ciência da Computação, no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Sfair Sunye

CURITIBA

2014

TARCIZIO ALEXANDRE BINI

ANÁLISE DA APLICABILIDADE DAS REGRAS DE OURO AO TUNING DE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

Tese apresentada como requisito parcial à obtenção do título de Doutor em Ciência da Computação, no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Sfair Sunye

CURITIBA

2014

Bini, Tarcizio Alexandre
Análise da aplicabilidade das regras de ouro ao tuning de sistemas gerenciadores de bancos de dados relacionais em ambientes de computação em nuvem / Tarcizio Alexandre Bini. – Curitiba, 2014.
97 f. : il., tabs., grafs.

Tese (doutorado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós Graduação em Informática
Orientador: Marcos Sfair Sunye
Bibliografia: p. 87-97

1. Banco de dados relacionais. 2. Banco de dados - Gerência 3. Computação em nuvem . I. Sunye, Marcos Sfair. III. Título.

CDD: 005.74

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa do aluno de Doutorado em Ciência da Computação, Tarcizio Alexandre Bini, avaliamos a tese de doutorado intitulada *"Analise da aplicabilidade das regras de ouro ao tuning de sistemas gerenciadores de bancos de dados relacionais em ambiente de computação em nuvem"*, cuja defesa pública foi realizada no dia 07 de março de 2014, às 14:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 07 de março de 2014.

Prof. Dr. Marcos Sfair Sunye
DINF/UFPR – Orientador

Prof. Dr. Clodis Boscarioli
UNIOESTE – Membro Externo

Prof. Dr. Gustavo Alberto Giménez Lugo
UTFPR – Membro Externo

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Membro Interno

Prof. Dr. Fabiano Silva
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço a Deus pelo seu imenso amor, me concedendo a vida, recheada de alegrias, tristezas, dificuldades, fracassos e sucessos, pelos seus inúmeros contrastes, que a tornam uma maravilhosa e infinita escola de progresso intelecto-moral. Sem Deus nada é possível.

Aos meus familiares, especialmente meus pais, Angelo e Sofia, por sempre incentivarem seus filhos aos estudos, também pela paciência e apoio nos momentos de dificuldades e dúvidas.

À minha namorada Débora, pelas palavras amigas e revigorantes, pelo seu sorriso no momento certo, pelo amor. Agradeço também pela ajuda nas correções deste documento.

A todos os professores do Departamento de Informática da Universidade Federal do Paraná. Agradeço especialmente ao meu orientador, Marcos Sfair Sunye, por acreditar em meu potencial me incentivando quando pensei em desistir. Obrigado pelas dicas e sugestões para o desenvolvimento e aprimoramento dos trabalhos.

Aos colegas de caminhada João Eugênio, Lucélia, Rebeca, Edson Ramiro e Cristiane (em memória), que nos primeiros anos do doutorado compartilharam de minhas dificuldades medos e incertezas. A frase "... se Deus quiser, um dia ainda acharemos graça de tudo isso ..." era sempre proferida.

Aos integrantes da Universidade Tecnológica Federal do Paraná (UTFPR) campus Guarapuava, professores, técnicos administrativos e terceirizados, que fizeram parte da história da etapa final deste curso de doutorado. Em especial ao grande amigo Eleandro Maschio pelo apoio decisivo em tantos momentos. Sucesso!

Agradeço ao acadêmico Andre Ziviani e aos professores Fabiano Silva e Marcos Castilho pela disponibilização do ambiente computacional e ajuda nas configurações necessárias à execução dos experimentos.

Finalmente, agradeço a todos aqueles que direta ou indiretamente, muitas vezes sem saber, contribuíram para o desenvolvimento e conclusão deste trabalho.

"Sabe, uma coisa mínima pode mudar sua vida. Num piscar de olhos alguma coisa acontece do nada, quando você menos espera e te coloca num caminho que você nunca planejou e um futuro que você nunca imaginou. Para onde ele vai te levar? É a jornada das nossas vidas, nossa busca pela luz. Mas, às vezes, para encontrar a luz você tem que passar pela mais profunda escuridão. Pelo menos, foi o que aconteceu comigo."

(Um Homem de Sorte, Nicholas Sparks)

RESUMO

A computação em nuvem oferece um ambiente bastante propício para o provimento de serviços de TI. A virtualização, tecnologia que compõe sua base possibilita simular sobre um computador físico, uma ou mais estações de trabalho chamadas *máquinas virtuais*, que permitem maior flexibilidade e melhor racionalização de sua infraestrutura. A incorporação de sistemas legados aos ambientes em nuvem como forma de contenção de custo é uma demanda frequente e altamente relevante. Para isso, é comum o emprego do modelo *multi-inquilino* do tipo *shared-hardware*, no qual o sistema gerenciador de banco de dados e o sistema legado ficam hospedados em máquinas virtuais que competem, junto às demais, por recursos computacionais. Neste ambiente, é vital o emprego de estratégias de *tuning* que objetivam melhorias no desempenho do banco de dados. Porém, os sistemas gerenciadores de banco de dados relacionais não foram inicialmente projetados para serem executados em ambientes *shared-hardware*. Consequentemente, seus parâmetros de configuração, comumente alvos de regras de *tuning*, não consideram o fato de que os recursos disponíveis variam ao longo do tempo, devido ao provisionamento dinâmico comum em ambientes elásticos. Esta tese propõe um método de avaliação que, por meio da simulação de cargas de trabalho de acesso a disco oriundas de máquinas virtuais concorrentes, demonstra a inadequação do emprego das regras de *tuning*, conhecidas como regras-de-ouro, encontradas na literatura e/ou recomendadas por *experts*. Nossos resultados apontam para a definição de novas regras-de-ouro, específicas para ambientes virtualizados, além de viabilizar a criação de um modelo para o *tuning* automático de sistemas gerenciadores de banco de dados relacionais em ambientes de computação em nuvem.

Palavras-Chave: sistema gerenciador de banco de dados relacional, virtualização, *tuning*, sistema legados, computação em nuvem.

ABSTRACT

Cloud computing currently offers a very propitious environment for IT service provision. The virtualization, technology that compose their base enables to simulate in a physical computer one or more workstations called virtual machines that allow greater flexibility and better use of its infrastructure. The incorporation of legacy systems to the cloud environments as a means of cost containment is a frequent and highly relevant demand. Therefore, it is common the use the multi-tenant model of shared-hardware type on which the database and legacy system are hosted on virtual machines that compete, with others, for computational resources. In this environment it is vital the use of tuning strategies that aim to improve the performance of the database. However, the relational database management systems were not initially designed to execute on shared-hardware environments. Consequently, its configuration parameters, commonly targets of tuning rules, do not consider the fact that the available resources vary over time due to the common dynamic provisioning that is common in elastic environments. This thesis proposes an evaluation methodology that, simulates I/O workloads from concurrent virtual machines and demonstrates the inadequacy of the use of tuning rules, known as rules-of-thumb, found in literature and/or recommended by experts. Our results point to the new rules-of-thumb, specific to virtualized environments while also make feasible the creation of a model for automatic tuning of database in cloud computing environments.

Keywords: relational database management system, virtualization, tuning, legacy systems, cloud computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ACRÔNIMOS

ACID - *Atomicity, Consistency, Isolation, and Durability*

AMD - *Advanced Micro Devices*

API - *Application Programming Interface*

AWS - *Amazon Web Services*

CPU - *Central Processing Unit*

CRM - *Customer Relationship Management*

DaaS - *Data as a Service*

DBaaS - *Database as a Service*

DSS - *Decision Support System*

GB - *Gigabyte*

Hz - *Gigahertz*

IaaS - *Infrastructure as a Service*

IDE - *Integrated Development Environment*

IP - *Infrastructure Provider*

KB - *KiloByte*

KVM - *Kernel-based Virtual Machine*

LIS - *Legacy Information System*

MB - *Megabyte*

MV - Máquina Virtual

MVVs - Monitor de Máquinas Virtuais

NIST - *National Institute of Standards and Technology*

OLAP - *Online Analytical Processing*

OLTP - *Online Transaction Processing*

OQL - *Object Query Language*

PaaS - *Plataform-as-a-Service*

QEP - *Query Execution Plan*

RAID - *Redundant Array of Independent Disks*

RAM - *Random Access Memory*

RPM - Rotações por Minuto

SaaS - *Software as a Service*

SATA - *Serial Advanced Technology Attachment*

SF - *Scale Factor*

SGBD - Sistema Gerenciador de Banco de Dados

SGBDR - Sistema Gerenciador de Banco de Dados Relacional

SO - Sistema Operacional

SP - *Service Provider*

SQL - *Structured Query Language*

TI - Tecnologia da Informação

TPC-H - *Transaction Processing Performance Council - Benchmark H*

XML - *eXtensible Markup Language*

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

A computação em nuvem tem se tornado um ambiente altamente propício para o provimento de serviços de TI (Tecnologia da Informação). O emprego massivo da virtualização, tecnologia que compõe sua base, não só têm proporcionado uma forma flexível e simples de administrar recursos computacionais, como também tem permitido reduções significativas de custos às organizações. Para isso, a virtualização busca simular a execução de várias estações de trabalho denominadas Máquinas Virtuais (MV) que compartilham entre si um conjunto de recursos computacionais, fornecidos por um computador físico chamado hospedeiro.

Embora novas técnicas de programação e arquiteturas de software tenham surgido especificamente para o paradigma de computação em nuvem, muitos sistemas legados ainda estão sendo mantidos em operação por diversas empresas. Representados por aplicações antigas e muitas vezes desatualizadas, os sitemas legados possuem o crítico encargo de manter adequadamente a regra de negócio das organizações. A incorporação desses sistemas à infraestrutura de computação em nuvem é uma necessidade frequente e relevante como forma de redução de custos operacionais, e a virtualização torna-se uma forma viável para esse propósito.

Muitos dos sistemas legados em operação foram projetados usando a tradicional arquitetura cliente/servidor. Tais sistemas são compostos principalmente por uma instância de um Sistema Gerenciador de Bancos de Dados Relacional (SGBDR) e por um programa cliente que executa na estação de trabalho do usuário, o qual necessita ter acesso ao SGBDR por meio de uma rede local. Uma boa alternativa para incorporar tais sistemas em nuvem é pelo uso de um software Monitor de Máquinas Virtuais (MMVs), criar uma MV para hospedar a instância de SGBDR e outra para hospedar a aplicação do cliente.

Muito embora o sistema legado também possa se encontrar hospedado na mesma MV que o SGBDR.

Mesmo a virtualização trazendo grandes benefícios administrativos e econômicos para a manutenção de sistemas legados, sua flexibilidade no provisionamento de recursos tem potencializado um problema crítico dos SGBDRs: a configuração e o ajuste de seu desempenho.

1.1 Definição do Problema

De um modo geral, existe um conflito importante entre maximizar o desempenho de um SGBDR e minimizar os recursos por ele utilizados. Tratando-se de ambientes de computação em nuvem de larga escala, tanto a subutilização de recursos computacionais como o super-dimensionamento de hardware são práticas indesejáveis, uma vez que aumentam de forma significativa os custos de infraestrutura repassados aos clientes. Para obter melhorias no desempenho das aplicações de banco de dados em ambientes virtualizados, uma dentre várias soluções que evitam elevados investimentos de recursos computacionais é a realização de intervenções nos mecanismos responsáveis pelo processamento de consultas.

Uma vez que os SGBDRs não foram inicialmente projetados para serem executados em ambientes virtualizados, seu modelo de custos, base para a tomada de decisões e otimizações, não leva em consideração sua execução em ambientes elásticos, que implicam em um provisionamento dinâmico de recursos. A existência de cargas de trabalho concorrentes, oriundas de outras MVs sobre um mesmo hardware, também é um exemplo que não se pode ignorar. É necessário que o mecanismo de auto-configuração do SGBDR reconheça essas variações que ficam de posse do MMVs, responsável pelo escalonamento de recursos. Para isso é preciso conceber uma nova arquitetura de custos para os SGBDRs.

A solução apresentada não é desejável quando o SGBDR é utilizado para o atendimento às requisições de sistemas legados, devido à necessidade de alterações no código-fonte do SGBDR, não suportadas pelos sistemas legados que muitas vezes são altamente depen-

dentes de versões específicas, comumente descontinuadas e obsoletas. Assim, uma solução menos invasiva objetivando melhorias no desempenho de SGBDRs inseridos em ambientes virtualizados é o emprego de regras de *tuning* encontradas na literatura e/ou recomendas por especialistas, também conhecidas como regras-de-ouro.

1.2 Hipótese de Pesquisa

Uma vez que as regras-de-ouro foram definidas para serem aplicadas à configuração de SGBDRs que executam em ambientes não-virtualizados, estas não consideram o efeito de cargas de trabalho concorrentes de acesso a disco, oriundas de outras MVs que executam sobre o mesmo hospedeiro. A necessidade de constantes acessos às unidades de disco, comuns em ambientes de produção, representam o maior "gargalo" para SGBDRs que manipulam grandes quantidades de dados. Assim, a hipótese principal desta tese é a de que as regras-de-ouro comumente utilizadas para o *tuning* de SGBDRs, não devem ser aplicadas quando os mesmos estiverem inseridos em ambientes virtualizados, sobre o ônus de não trazer benefícios ou até mesmo prejudicar drasticamente seu desempenho.

1.3 Objetivos e Contribuições

Para que nossa hipótese pudesse ser comprovada foi necessária a concepção de um método para avaliar os efeitos do emprego de regras de *tuning* sobre SGBDRs que operam sobre ambientes virtualizados. Para isso, o PostgreSQL [PostgreSQL, 2013] foi instalado em uma MV recebendo de forma concorrente à sua execução, intenso acesso às unidades de armazenamento provenientes de outras MVs. Destaca-se que as MVs estavam alocadas sobre o mesmo hospedeiro.

A carga de trabalho de acesso a disco foi disparada por processos resultantes da adaptação de um *benchmark* que analisava o desempenho e o comportamento de discos rígidos e sistemas de arquivos. Uma vez que os discos são dispositivos mecânicos, tais cargas de trabalho são caracterizadas e implementadas considerando em uma dimensão

requisições de *Leitura* e *Escrita*, e em outra, acessos *Aleatórios* e *Sequenciais* ao disco.

Foi necessária a elaboração de um conjunto de consultas com características específicas, que simultaneamente provocam intensas requisições a disco e também oportunizam analisar os efeitos das regras de *tuning* sobre o total de parâmetros de configuração escolhidos. Tais consultas tomam por base as que foram fornecidas por um *benchmark* que proveu a base de dados sintética que simula um ambiente análitico e de suporte a decisões utilizado nos experimentos.

Os parâmetros considerados na análise experimental foram escolhidos baseados em classificações encontradas na literatura. Estas classificações apontam os parâmetros que causam maior efeito no desempenho do SGBDR considerando a carga de trabalho de consultas submetida e as características da base de dados sintética utilizada. Evitou-se assim, o aumento exponencial na quantidade de resultados obtidos.

O método implementado e descrito nesta tese demonstra a inadequação do emprego de regras-de-ouro ao *tuning* de SGBDR inseridos em ambientes virtualizados. Também está apto a quantificar tal inadequação quando comparada às demais regras de *tuning* experimentadas. Para isso, é considerado o tempo médio de execução de cada consulta e também o tempo médio para execução da carga de trabalho composta por todas as consultas elaboradas. Nosso método exibe os tempos médios de execução contemplando cada um dos 4 tipos de acesso a disco elencados.

Os resultados obtidos pelo método que são exibidos na forma de gráficos, também apontam para a definição de novas regras-de-ouro para a configuração adequada de SGBDRs em ambientes virtualizados. Tais regras apresentam sugestões de valores a serem alocados à cada um dos parâmetros de configuração analisados considerando também cada tipo de acesso a disco. Na maioria dos casos os valores sugeridos estão bem distantes dos propostos por especialistas ou pela literatura que considera a execução do SGBDR em ambiente não virtualizado.

Por fim, nossos resultados viabilizam o desenvolvimento de um modelo para realizar o *tuning* automático dos SGBDRs em ambientes virtualizados. Para isso, as novas regras-

de-ouro apontadas por este trabalho devem ser adotadas simultaneamente, de acordo com a análise das características do tipo da carga de acesso a disco, oriundas de outras MVs, que executam concorrentemente ao SGBDR. Esta tese também mensura diversos trabalhos futuros relativos a otimização de desempenho de SGBDRs que apresentam importância e relativa urgência em desenvolvimento.

1.4 Organização do Documento

Esta tese encontra-se assim organizada:

No Capítulo 2 estão as definições dos conceitos e tecnologias relacionadas ao ambiente de computação em nuvem. A descrição principal do Capítulo é a respeito da tecnologia de virtualização assim como seus tipos.

No Capítulo 3 é colocada em foco a inserção de Sistemas Gerenciadores de Banco Dados (SGBD) em ambientes de computação em nuvem considerando a tecnologia de virtualização. Também é apresentada a classificação dos mesmos dando atenção especial aos SGBDRs.

Cabe ao Capítulo 4 realizar discussões sobre duas técnicas de otimização de desempenho de SGBDRs, uma relacionada ao processamento de consultas e outra relacionada à utilização de regras de *tuning*. Descreve-se no Capítulo 5 de forma detalhada o ambiente computacional empregado na implantação de nosso método.

No Capítulo 6 estão os resultados obtidos através dos experimentos e considerações a respeito do emprego de regras de *tuning* para a otimização de desempenho de SGBDRs, sobre MVs que recebem acesso concorrente a disco.

Apresentam-se no Capítulo 7, as conclusões alcançadas, bem como os trabalhos futuros.

Os Apêndices A e B encerram esta tese, apresentando respectivamente as consultas utilizadas em nossos experimentos e os trabalhos desenvolvidos sobre a otimização de desempenho de SGBDRs.

CAPÍTULO 2

COMPUTAÇÃO EM NUVEM: CONCEITOS E TECNOLOGIAS

A constante exigência por recursos computacionais demandou num passado não muito distante o desenvolvimento da computação distribuída e paralela. Em seguida, pesquisas relacionadas a *clusters* e, posteriormente, grades computacionais alcançaram êxito, fazendo com que seu emprego em aplicações reais se tornasse um sucesso. Atualmente o modelo de computação em nuvem ou *cloud computing* [Mc Evoy et al., 2011, Zhang et al., 2010, Wang et al., 2010, Vouk, 2008] é uma tendência adotada pela indústria, governo e comunidade científica como solução distribuída, flexível e elástica.

Ainda não havendo um consenso sobre a definição do termo computação em nuvem, um conceito mais amplo e não definitivo, pois se encontra em constante evolução, é fornecido pelo (NIST) [Peter Mell and Timothy Grance, 2011] *National Institute of Standards and Technology*: *"cloud computing é um modelo que permite o acesso, de forma conveniente e sob demanda a recursos configuráveis de computação, como, armazenamento, processamento e aplicações. Estes, podem ser rapidamente adquiridos e liberados com o mínimo de esforço de gestão ou interação com o provedor de serviços".*

Considerando tal conceito, constata-se que um dos principais objetivos da computação em nuvem é fornecer a seus usuários a pilha computacional sob a forma de serviços, com pagamento baseado na utilização (*pay-per-use*), [Buyya et al., 2009]. Assim, evita-se o grande investimento de capital para construção, aquisição e instalação de equipamentos de computação em larga escala. Como o acesso aos serviços é realizado de forma simples e transparente, não há necessidade dos usuários conhecerem previamente as tecnologias utilizadas, tão pouco a forma da implementação física da nuvem. Para que o usuário possa usufruir dos serviços oferecidos, é necessário no mínimo que sua máquina tenha

acesso à Internet e que previamente tenham sido instalados um sistema operacional e um aplicativo navegador.

No cenário descrito, o modelo de computação em nuvem é composto basicamente por três (3) atores principais [Vaquero et al., 2008]: (1) **provedores de infraestrutura** ou IP's (*Infrastructure Providers*), que fornecem serviços computacionais e de armazenamento, necessários à execução de aplicações dentro do modelo; (2) **provedores de serviços** ou SP's (*Services Providers*), responsáveis pela disponibilização, gerenciamento e monitoramento de serviços. Estes, na maioria das vezes desenvolvem aplicações que são oferecidas e implantadas na plataforma de computação em nuvem; e os (3) **usuários de serviços**, que utilizam os recursos fornecidos pela nuvem computacional, obtendo-os de diversos canais como de provedores de serviço e infraestrutura.

Segundo o NIST [Peter Mell and Timothy Grance, 2011], tipicamente o modelo de computação em nuvem é composto por cinco (5) características principais:

- **Serviço sob demanda**: os usuários podem obter recursos computacionais como processamento, armazenamento de forma automática, sem a interação humana com os provedores de serviços.

- **Amplo acesso à rede**: os recursos computacionais são disponibilizados através da rede e acessados por meio de mecanismos padronizados que possibilitam o uso por dispositivos como celulares, *notebooks* e estações de trabalho.

- **Agrupamento de recursos**: os recursos computacionais fornecidos pelos provedores de serviços são agrupados de forma a atender múltiplos usuários, com diferentes recursos físicos e virtuais, dinamicamente ajustados e atribuídos conforme demanda.

- **Elasticidade**: Os recursos computacionais podem ser adquiridos ou liberados de forma rápida, elástica e em alguns casos automaticamente. Para o usuário, os recursos parecem ser ilimitados podendo ser requisitados a qualquer momento e em qualquer quantidade.

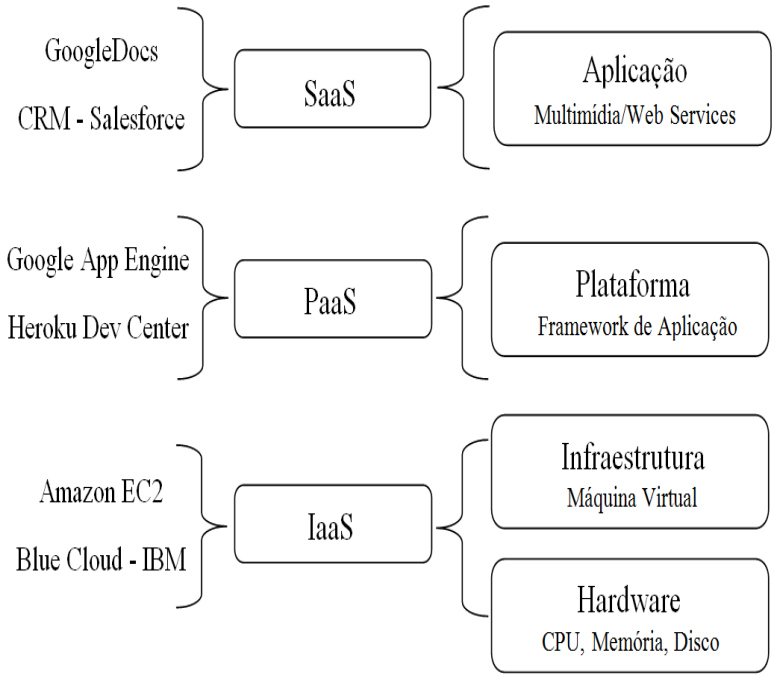


FIGURA 2.1: OS TRÊS MODELOS DE SERVIÇOS DA COMPUTAÇÃO EM NUVEM

- **Monitoramento dos serviços**: Os sistemas de gerenciamento de computação em nuvem controlam e otimizam automaticamente os recursos computacionais para cada tipo de serviço (processamento, armazenamento e largura de banda). Tal monitoramento de recursos, deve ser transparente tanto para o provedor quanto para o usuário dos serviços utilizados.

2.1 Modelos de Serviços de Computação em Nuvem

Os serviços oferecidos pelo ambiente de computação em nuvem são classificados com maior frequência na literatura em três (3) modelos [Rimal et al., 2009, Chieu et al., 2009]. Esta classificação é importante, uma vez que define o padrão arquitetural para as soluções de computação em nuvem. Estes modelos provêem níveis variáveis de economia e dependência dos provedores de serviços, no que diz respeito à integração entre os vários sistemas locais de uma empresa e os sistemas a serem implantados em nuvem. A Figura 2.1 busca sintetizar os três (3) modelos que serão discutidos, dando exemplos de serviços oferecidos por cada um.

Software como Serviço: Este modelo, também conhecido pela sigla SaaS (*Software as a Service*), provê aos usuários através da Internet e de uma interface *thin client*, sistemas de software com finalidades específicas. O usuário não necessita controlar ou administrar a infraestrutura subjacente da nuvem que pode incluir entre outros, rede, sistemas operacionais e armazenamento. Isso resulta em um desenvolvimento mais rápido de sistemas de software, além da redução nos custos relacionados à aquisição de licenças, atualizações de software e *backup* de dados que ficam sob responsabilidade do provedor.

O modelo SaaS, representa os serviços de mais alto nível providos pela computação em nuvem e exige pouca ou nenhuma alteração na infraestrutura das organizações. Tem como exemplo de representantes o *GoogleDocs* e o serviço de gestão do relacionamento com o cliente (*Customer Relationship Management*-CRM) da Salesforce [CRM Salesforce, 2013].

Plataforma como Serviço: Também conhecido como PaaS (*Plataform-as-a-Service*), este modelo de serviço fornece aos seus usuários uma plataforma para o desenvolvimento e teste de aplicações sobre a nuvem computacional, com foco na colaboração entre os desenvolvedores. Semelhante ao modelo SaaS, o usuário não necessita gerenciar a infraestrutura básica da nuvem. De acordo com a disponibilidade do provedor de serviços, o usuário poderá, por exemplo, ter a sua disposição compiladores, IDEs (*Integrated Development Environment*), ferramentas para controle e gerenciamento de versões, ferramentas para testes de desempenho e testes automatizados. Como exemplo de PaaS pode-se citar o *Google App Engine* [Google App Engine, 2013] e o *Heroku Dev Center* [Heroku, 2013].

Infraestrutura como Serviço: O terceiro modelo de serviços também denominado IaaS (*Infrastructure as a Service*) fornece aos usuários da nuvem recursos computacionais. Mesmo não controlando ou administrando a infraestrutura da nuvem é permitida ao usuário a escolha do sistema operacional, da quantidade de memória, disco e núcleos de processamento que fará uso, além de controle limitado de componentes como *firewalls*. Esta infraestrutura é baseada na tecnologia de *virtualização*, que permite escalar dinamicamente os recursos computacionais, aumentando ou diminuindo sua disponibilidade às aplicações, conforme a necessidade dos usuários. Tal tecnologia que oportuniza

fornecer aos usuários a infraestrutura computacional sobre a forma de serviços é de extrema importância à computação em nuvem e será detalhada posteriormente, em subseção específica.

O modelo IaaS permite que as organizações reduzam consideravelmente os custos relacionados a hardware e seu suporte, além da economia em energia e melhor aproveitamento de espaço físico. São exemplos de provedores deste modelo o *Amazon Elastic Cloud Computing* (EC2) [ama, 2013] e o *Blue Cloud da IBM* [Blue Cloud, 2013].

Uma vez que a computação em nuvem fornece a ideia de que as necessidades dos usuários são providas como serviços, muitas de suas soluções deram origem a termos que passaram a ser conhecidos como novos modelos de serviços. Ao contrário dos três já apresentados, existe pouca consensualidade na literatura sobre suas definições, como é o caso do modelo DaaS (*Data as a Service*) ou DBaaS (*Database as a Service*) [Rodrigues, 2013]. Neste, o objetivo é disponibilizar a partir dos dados referentes a determinado negócio, análises e estudos de tendências sobre um contexto em específico, de forma segura, rápida, com custos reduzidos e acessíveis através da Internet. Também existem soluções que visam disponibilizar e implantar bases de dados sobre o modelo de computação em nuvem, oferecendo espaço de armazenamento, além de todas as funcionalidades e serviços inerentes à gestão de dados.

2.2 Modelos de Implantação de Computação em Nuvem

Considerando as características de implantação da nuvem computacional é possível a identificação de quatro (4) modelos segundo NIST [Peter Mell and Timothy Grance, 2011]. Estes modelos definirão a disponibilidade e as restrições de acesso aos serviços oferecidos aos usuários.

Nuvem Privada: neste modelo, a infraestrutura de nuvem é utilizada exclusivamente por determinada organização, com usuários devidamente autorizados. Local ou remota, pode ser administrada pela própria organização (o que mais comumente ocorre) ou por

terceiros. Neste âmbito, políticas de acesso são empregadas considerando tecnologias de autenticação, autorização e gerenciamento de redes.

As nuvens privadas permitem a melhor alocação de recursos de forma a atender as necessidades individuais da organização. Também, torna-se uma solução atrativa para instituições que procuram otimizar grandes investimentos em TI já realizados. Os recursos do ambiente computacional, como servidores, *desktops* e *storages* são agrupados, centralizados e oferecidos aos usuários como serviços, conforme o modelo IaaS.

Nuvem Pública: a infraestrutura provida pela nuvem é disponibilizada a usuários de forma geral, os quais devem conhecer a localização do serviço oferecido. Geralmente, nuvens públicas são executadas por terceiros tendo seus serviços gratuitos ou comercializados. Este modelo de nuvem é uma boa solução para empresas que necessitam executar determinada tarefa de forma flexível e temporária garantindo a redução de custos. Como principais benefícios deste modelo, estão a facilidade de configuração e utilização dos serviços, a escalabilidade e o pagamento conforme uso, evitando assim despender grandes investimentos em infraestrutura de TI.

Apesar das vantagens oferecidas, muitas organizações não se sentem confortáveis em utilizar o modelo de nuvem pública para seus sistemas críticos. Incertezas surgem a respeito da privacidade e segurança das informações, compartilhadas com outras organizações, muitas vezes localizadas em outros países. Para estes casos, o modelo de nuvem híbrida pode ser mais atrativo.

Nuvem Comunidade: a infraestrutura da nuvem é compartilhada por uma comunidade, ou seja, um conjunto de organizações que partilham objetivos, preocupações e interesses em comum, por exemplo, requisitos de segurança, informações, ferramentas e aplicativos específicos. A nuvem pode estar posicionada localmente ou remotamente sendo gerenciada, operada e controlada por uma ou mais organizações que compõem a comunidade ou por terceiros.

Nuvem Híbrida: neste modelo de implantação existe uma composição de nuvens, podendo ser públicas, privadas ou comunidades. Cada entidade permanece única sendo

interligada por tecnologias padronizadas, as quais permitem a portabilidade de dados e aplicações.

O modelo de nuvem híbrida permite que uma organização gerencie sua infraestrutura de TI, seu uso sob a forma de serviços e também obtenha recursos externos de terceiros. Esta abordagem possibilita tirar proveito da escalabilidade e contenção de custos que a nuvem pública oferece, não expondo, pelo modelo de nuvem privada, dados e aplicações críticas à vulnerabilidade de terceiros. Assim, é possível que uma nuvem privada tenha seus recursos potencializados a partir da reserva de recursos oferecidos por uma rede pública, mantendo os níveis de serviço mesmo na ocorrência de flutuações na disponibilidade de recursos.

2.3 A Virtualização e a Computação em Nuvem

A computação em nuvem é constituída por diversas tecnologias, como a virtualização que compõe a sua base [Zhang et al., 2010]. Em uma visão ampla, a virtualização é a simulação através de software, de uma ou mais estações de trabalho ou servidores em um computador físico, o que permite que um único computador desempenhe o papel de vários, partilhando seus recursos por meio da multiplicidade de ambientes.

Um sistema virtualizado consiste basicamente de três (3) partes [Maziero, 2013]: (1) o **hospedeiro** (*host system*) que possui recursos reais de hardware e software do sistema; (2) a **Máquina Virtual** (MV), também denominada sistema convidado (*guest system*) que executa sobre o sistema virtualizado, podendo em muitos casos coexistir com várias MVs, executando simultaneamente sobre o mesmo hospedeiro; (3) a **camada de virtualização**, denominada *hipervisor* ou *Monitor de Máquinas Virtuais* (MMVs) que permite o suporte e gestão de MVs sobre o mesmo hospedeiro.

Semelhante a uma máquina física, cada MV irá fornecer a seu utilizador um ambiente completo, podendo ter entre outros, seu próprio sistema operacional, conjunto de aplicações, serviços de rede, processamento e armazenamento. Dessa forma, a virtua-

lização permite a divisão de recursos computacionais fazendo com que as aplicações sejam executadas de forma isolada a partir de uma plataforma de hardware compartilhada. Tal divisão é uma solução atrativa que resulta em redução de custos operacionais e de gestão [Rose, 2004].

O MMVs é responsável pelo controle e gestão dos recursos computacionais compartilhados pelo hospedeiro como memória, disco rígido, processamento e dispositivos de Entrada/Saída. De acordo com as necessidades, estes recursos podem ser disponibilizados às MVs pelos usuários autorizados ou de forma automática por aplicativos específicos. Dessa forma, servidores, dispositivos de armazenamento e outros hardwares são tratados como um conjunto de recursos que podem ser alocados conforme demanda [Chieu et al., 2009]. Para isso o MMVs implementa uma camada de software com o objetivo de realizar o tratamento e o escalonamento de instruções das MVs, fornecendo ao sistema operacional convidado a abstração da MV.

Segundo Chieu *et al.* [Chieu et al., 2009] e a VMWare [VMWARE, 2011], a tecnologia de virtualização é capaz de apresentar soluções aos problemas de gerenciamento de grandes *data centers*, além de oferecer benefícios à médias e pequenas empresas como:

- Redução de custos operacionais;
- Redução do tempo despendido com rotinas administrativas de TI;
- Facilidade na realização de *backup* e proteção de dados;
- Consolidação de recursos de TI (hardware/software/dados);
- Suporte a aplicações e hardware legado;
- Melhor disponibilidade de aplicativos;
- Facilidade de recuperação de falhas;
- Melhoria na escalabilidade dos sistemas computacionais;
- Facilidade de experimentação envolvendo redes e sistemas distribuídos.

2.3.1 Tipos de Virtualização

Diferentes abordagens para implementação de virtualização em servidores foram propostas, sendo as mais utilizadas discutidas a seguir:

Virtualização Total: Também chamado de *Full Virtualization* ou *Virtualização Completa*, provê uma réplica virtual do hardware subjacente. Como a estrutura completa de hardware é virtualizada, o sistema operacional convidado não necessita de modificações para suportar a virtualização, o que representa uma vantagem dessa abordagem. Porém, alguns inconvenientes merecem destaque. Uma vez que um computador apresenta grande diversidade de dispositivos de hardware, há dificuldade em implementar uma MV que simule o comportamento exato de cada um. Assim, torna-se necessário o emprego de *drivers* genéricos que muitas vezes inibem o uso da capacidade total do dispositivo. Outro inconveniente está relacionado à exigência do MMVs traduzir e analisar as instruções executadas pelo sistema convidado em uma ou mais instruções equivalente no hardware real, o que representa um custo extra de processamento [Maziero, 2013, Carissimi, 2008]. *VMWare Server* [VMWare, 2013] e o *Virtual Box* [Virtual Box, 2013] são exemplos de MMVs que se utilizam deste tipo de virtualização, que é ilustrado na Figura 2.2.

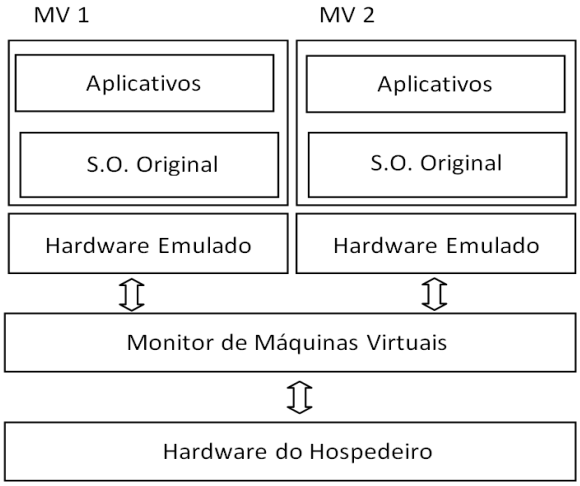


FIGURA 2.2: VIRTUALIZAÇÃO TOTAL

Paravirtualização: Esta abordagem busca contornar as inconveniências encontradas na virtualização total, permitindo que o sistema operacional convidado tenha acesso direto

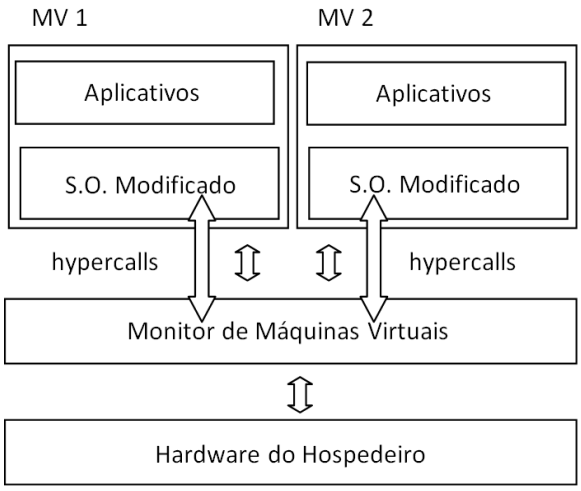


FIGURA 2.3: PARAVIRTUALIZAÇÃO

aos recursos de hardware, porém, com restrições administradas pelo MMVs. Os dispositivos de hardware são acessados pelos *drivers* próprios da MV não havendo necessidade de *drivers* genéricos [Peixoto, 2012]. O *kernel* do sistema operacional convidado é modificado permitindo que o mesmo faça chamadas (*hypercalls*) ao MMVs. A partir dessas chamadas, o MMVs realiza tarefas críticas como gestão de memória ou interrupções, em nome do *kernel* do sistema operacional da MV, que tem ciência de estar sendo executado em um ambiente virtual. Esta solução que é ilustrada na Figura 2.3, permite melhor desempenho comparada a virtualização total, uma vez que a análise e tradução de instruções não são necessárias. Como exemplo de MMVs que dão suporte a paravirtualização, pode-se citar o *Xen* [Xen, 2013], e o *IBM z/VM* [IBM z/VM, 2013].

Virtualização em Nível do Sistema Operacional: Nesta abordagem, também chamada de *OS-level virtualization*, a virtualização do servidor físico é realizada em nível de sistema operacional, permitindo assim a execução de diversas MVs de forma isolada e segura. Para isso, instala-se uma camada de software de virtualização sobre o sistema operacional do hospedeiro (sistema operacional anfitrião), permitindo que as MVs o compartilhem. Assim, o mesmo núcleo de sistema operacional é utilizado (geralmente uma instância do mesmo) para implementar o ambiente dos sistemas convidados, tendo cada um deles seus próprios recursos e sistema de arquivos, funcionando isoladamente.

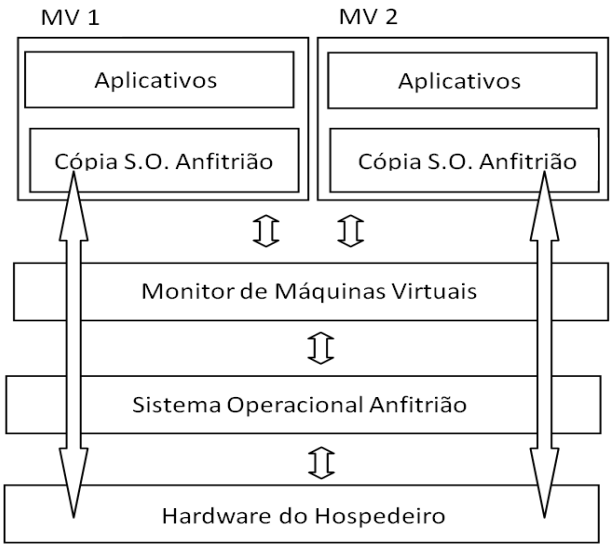


FIGURA 2.4: VIRTUALIZAÇÃO EM NÍVEL DO SISTEMA OPERACIONAL

A funcionalidade do MMVs é limitada, uma vez que conta com o sistema operacional hospedeiro para realizar o escalonamento de CPU e o gerenciamento de memória. A principal desvantagem desta abordagem de virtualização é que o usuário não pode fazer uso de outro sistema operacional nas MVs diferente daquele instalado no hospedeiro. A Figura 2.4 ilustra esta abordagem de virtualização, sendo exemplos de MMVs que a suportam, o *Solaris Zones* [Solaris Zones, 2013] e o *KVM* (*Kernel-based Virtual Machine*) [Kvm, 2013].

Virtualização Assistida por Hardware: ou *hardware-assisted virtualization*, esta abordagem faz uso de novas tecnologias de processamento como a Intel-VT e a AMD-V, as quais incorporam funcionalidades de suporte a virtualização. Tais processadores fornecem um modo de privilégio adicional no qual o MMVs pode operar. Assim, são providas extensões para a execução de MVs sem a exigência de alterações em seus sistemas operacionais e sem a necessidade de análise e tradução das instruções do sistema convidado para execução sobre o hardware real, conforme ilustra a Figura 2.5. Como exemplo de MMVs com suporte a este tipo de virtualização temos o *VMware* [VMWare, 2013], o *KVM* [Kvm, 2013] e o *Xen* [Xen, 2013].

Segundo Chieu *et al.* [Chieu et al., 2009], a virtualização oferece inúmeros benefícios, porém também traz desvantagens principalmente no que se refere à sobrecarga de de-

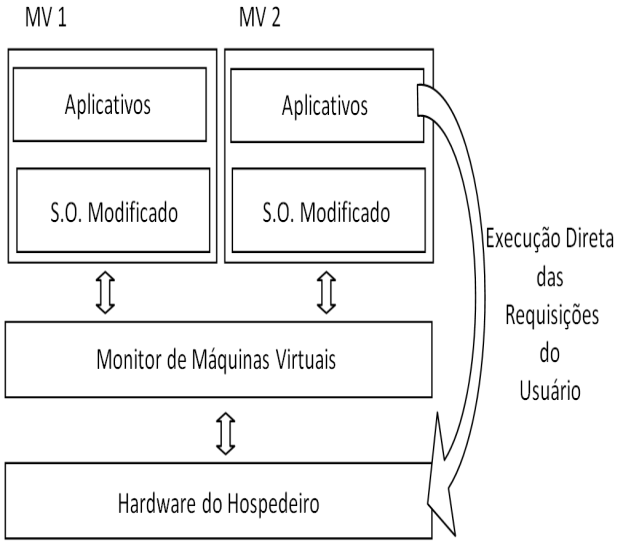


FIGURA 2.5: VIRTUALIZAÇÃO ASSISTIDA POR HARDWARE

sempenho. Isto se deve ao fato do MMVs atuar como componente de núcleo do sistema, funcionando como uma camada de software entre o hardware real e o sistema operacional. Esta camada consome recursos extras como processamento e memória quando comparada a um ambiente sem virtualização. Outra desvantagem relacionada refere-se à dificuldade de isolamento entre as MVs, principalmente no que diz respeito à utilização de recursos como disco e processamento, os quais sofrem constantes acessos de forma concorrente entre as MVs. Isto representa o principal "gargalo" de desempenho para os sistemas virtualizados.

Finalizamos nossa discussão sobre virtualização, afirmando que, independente da escolha de seu tipo, uma questão é certa, esta tecnologia ampliou os limites com que a capacidade computacional era empregada, permitindo que recursos (infraestrutura, plataforma e aplicações) sejam adquiridos como serviços conforme demanda. Tornando a computação em nuvem factível, a virtualização oportuniza à detentores de grandes *data centers* empregar suas capacidades computacionais ociosas como forma de fornecer serviços a diversos tipos de consumidores, reduzindo prejuízos com a subutilização da infraestrutura [Nobile, 2013].

CAPÍTULO 3

GESTÃO DE DADOS EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

Conforme discutido no Capítulo 2, o modelo de computação em nuvem vem alterando a forma como as soluções de tecnologia são acessadas e consumidas pelos usuários. A inserção de SGBDs neste modelo é uma demanda das organizações, que exigem a gestão segura e eficiente de quantidades crescentes de informações e, nos últimos anos, os serviços de gestão e armazenamento de dados em nuvem (DaaS e DBaaS) tornaram-se extremamente atrativos [Curino et al., 2011], como forma de redução de custos operacionais. Este modelo de serviços tem interessado à clientes de diversos setores do mercado, desde pequenas e médias empresas com o objetivo de reduzir custos através da utilização da infraestrutura de terceiros, até grandes empresas na busca de soluções à gestão de grandes volumes de dados e atendimento ao aumento na quantidade de requisições de forma segura e escalável.

São diversas as arquiteturas e sistemas baseados no modelo de computação em nuvem em desenvolvimento, que buscam suprir a demanda de aplicações que possuem diferentes necessidades de armazenamento e processamento. Tais sistemas fornecem aos usuários a visão de escalabilidade, processamento e armazenamento infinitos. Porém, enfrentam o problema de provisionamento de recursos [Sousa et al., 2011]. Neste contexto, a Tabela 3.1 apresenta, segundo [Curino et al., 2011], um conjunto de requerimentos chave para os serviços de gestão e armazenamento de dados em nuvem a partir da perspectiva do usuário, do provedor de serviços e requisitos adicionais relacionados à nuvem pública.

Considerando a perspectiva do usuário, sua principal necessidade está relacionada a um serviço de banco de dados oferecido por meio de uma interface simples, que não necessite de administração ou ajustes, contrastando assim, com as tradicionais soluções que requerem

TABELA 3.1: REQUISITOS PARA BANCOS DE DADOS COMO SERVIÇOS

Requisitos do Usuário	
U1	API simples, com pouca adminsitração e configuração (exemplo sem *tuning*)
U2	Alto desempenho e escalabilidade
U3	Alta confiança e disponibilidade (exemplo *backup*)
U4	Acesso fácil à característica avançadas (exemplo mineração de dados)
Requisitos do Provedor	
P1	Atender ao acordo de nível de serviço do usuário (sob carga dinâmica)
P2	Limitar custos de hardware e energia (exemplo multiplexação intensiva)
P3	Limitar custos de administração (exemplo custos com pessoal)
Requisitos da Nuvem Pública	
N1	Esquema de preços: barato, previsivel e proporcional ao uso (*pay-per-user*)
N2	Garantias de segurança e privacidade
N3	Baixa latência (relevante para aplicações web e transacionais)

FONTE: Modificado pelo autor [Curino et al., 2011]

técnicas para provisionar recursos, instalar, configurar e administrar SGBDs. Os usuários também demandam desempenho satisfatório, independente de alterações nas cargas de trabalho e tamanho da base de dados. Outro requisito é a alta disponibilidade, oferecida pelos SGBDs tradicionais, mas que requerem configurações e manutenções. Por fim, características avançadas de gerenciamento relacionadas a mineração de dados devem estar disponíveis e serem fáceis de utilizar [Curino et al., 2011].

Na perspectiva do provedor de serviços, é essencial o atendimento aos níveis de serviços acordados, independente da quantidade de dados e das mudanças nas cargas de trabalho. Para estas atividades deve-se manter um bom desempenho, utilizando recursos de hardware de forma eficiente. Um exemplo é a multiplexação, a capacidade de executar diversas MVs sobre o mesmo hospedeiro que resulta em melhor aproveitamento dos recursos computacionais disponíveis. Finalmente, a quantidade de atividades administrativas deve ser minimizada, utilizando ferramentas sofisticadas para a análise de cargas de trabalho, além da centralização de gerenciamento dos diversos bancos de dados [Curino et al., 2011].

Considerando os provedores de serviços em nuvem pública, requisitos como esquemas de preços, latência, segurança e privacidade devem ser contemplados. Porém, como não são questões específicas de bancos de dados, ficam fora do escopo deste trabalho.

3.1 Bancos de Dados Multi-Inquilino

Os provedores *PaaS* que encontram no modelo de gestão e armazenamento de dados em nuvem um de seus mais significantes componentes, deparam-se constantemente com o desafio de gerenciar e armazenar dados oriundos de centenas, ou mesmo milhares de pequenas aplicações chamadas *tenants* ou *inquilinos*. Dedicar um servidor de banco de dados para cada inquilino resulta em desperdício de recursos computacionais, uma vez que suas requisições individuais em maioria são pequenas. Uma solução que vem sendo amplamente adotada como forma de consolidação de recursos empresariais, são os bancos de dados *multitenant* ou *multi-inquilinos*. Esta técnica permite aos SGBDs em nuvem gerenciar grande número de inquilinos com padrões de carga de trabalhos irregulares. Assim, as aplicações de múltiplos usuários são consolidadas em um único sistema, evitando a necessidade de sistemas separados para cada inquilino.

Devido ao seu potencial, ao longo dos anos, os bancos de dados multi-inquilinos, tem despertado a atenção de pesquisadores, dentre eles [Curino et al., 2011, Hui et al., 2009, Aulbach et al., 2008, Jacobs et al., 2007]. Assim, vários modelos de implementação tem sido propostos e avaliados, sendo os principais: *shared-table*, *shared-process* e *shared-hardware*, detalhados a seguir. Tal classificação varia de acordo com o nível de abstração e isolamento [1]. [Elmore et al., 2011] propõem a Tabela 3.2, que demonstra o grau de isolamento entre os inquilinos e o seu correspondente paradigma de computação em nuvem, devidamente descrito no Capítulo 2 desse trabalho.

Shared-Table: este modelo que é ilustrado na Figura 3.1, compartilha um único conjunto de tabelas entre os inquilinos. Cada tabela possui uma coluna (*tenantid*), que define a qual inquilino cada linha pertence. Uma seleção considerando um identificador específico nesta coluna, irá recuperar somente as linhas pertencentes a determinado inquilino.

Todos os inquilinos estarão compartilhando o mesmo esquema das tabelas. Porém, é possível que o mesmo estenda seu esquema lógico de acordo com suas necessidades. Assim,

[1] Isolamento no contexto multi-inquilino está relacionado ao isolamento de desempenho, recursos e acessos entre os inquilinos que compartilham o sistema.

TABELA 3.2: MODELOS DE BANCOS DE DADOS MULTI-INQUILINOS E A CORRESPONDÊNCIA COM A COMPUTAÇÃO EM NUVEM

Modo de Compartilhamento	**Isolamento**	**IaaS**	**PaaS**	**SaaS**
Hardware	Máquina Virtual	X		
Máquina Virtual	Usuário Sist. Oper.		X	
Sistema Operacional	Instância Banco de Dados		X	
Instância	Esquema		X	
Tabela	Linha			X

FONTE:: [Elmore et al., 2011]

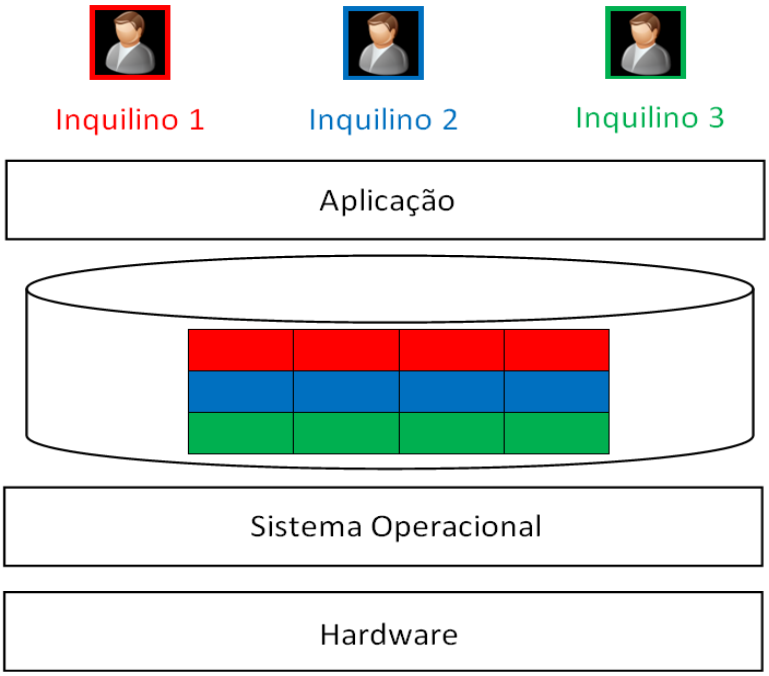


FIGURA 3.1: MODELO MULTI-INQUILINO *SHARED-TABLE*
FONTE: Modificado pelo autor [Agrawal et al., 2012]

lhe é permitido, por exemplo, acrescentar e renomear atributos sem afetar o esquema lógico de outros inquilinos. Além da extensão de esquema, outros requisitos como estatísticas, rotinas de *backup* e recuperação por inquilino, além da necessidade de migração entre máquinas devem ser suportadas pelo sistema.

Mesmo oferecendo a vantagem de manter uma única instância de base de dados, tal característica pode limitar funcionalidades específicas de cada tipo de SGBD, pois obriga o provedor de serviços a utilizar a mesma versão para todos os inquilinos. Outro problema está relacionado ao isolamento de inquilinos para migração de sistemas, devido aos meca-

nismos de bloqueios implementados. O modelo *shared-table* é ideal quando os requisitos de dados dos inquilinos seguem estruturas e padrões semelhantes. Um exemplo de destaque é o *Customer Relationship Management* da *Salesforce.com* [CRM Salesforce, 2013], que oferece customizações a seus usuários e já conseguiu consolidar cerca de 17 mil inquilinos em um banco de dados [Schiller et al., 2011]. Isto se deve principalmente ao fato deste modelo manter um consumo de memória principal constante, conforme o número de inquilinos aumenta.

Shared-Process: Os inquilinos compartilham uma única instância do banco de dados, porém, cada inquilino obtém um conjunto privado de tabelas, conforme ilustrado na Figura 3.2. Este modelo permite o efetivo compartilhamento de recursos entre os inquilinos e permite ao sistema de banco de dados gerenciar de forma inteligente recursos críticos como discos, o que possibilita que um maior número de inquilinos sejam consolidados no mesmo servidor mantendo bom desempenho.

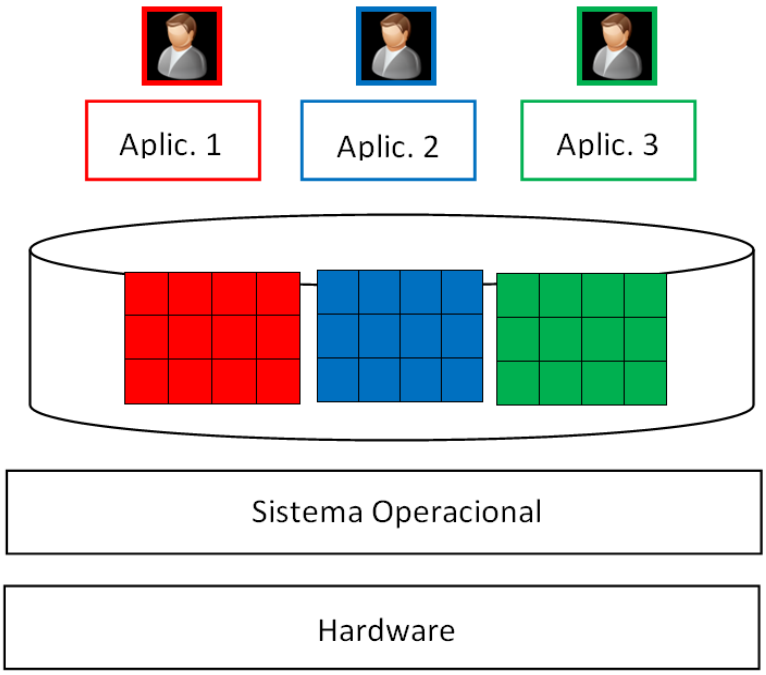


FIGURA 3.2: MODELO MULTI-INQUILINO *SHARED-PROCESS*
FONTE: Modificado pelo autor [Agrawal et al., 2012]

O modelo *shared-process* consome menos memória principal por inquilino. Em contrapartida, seu consumo aumenta rapidamente conforme os inquilinos individualmente

obtêm uma instância dedicada do esquema [Schiller et al., 2011]. Este modelo é apropriado quando se é necessário que uma aplicação sirva a um grande número de inquilinos, mesmo havendo um número reduzido de servidores. Tal aplicação deve ter um pequeno número de tabelas por inquilino. Seus usuários devem estar cientes que o isolamento dos dados não será tão eficiente como na abordagem *shared-hardware*, uma vez que estarão co-alocados com os dados de outros clientes. Em compensação, os serviços serão oferecidos a custos mais baixos. Esta característica torna esse modelo interessante para pequenas e médias empresas. O modelo *shared-process* pode ser observado em provedores *PaaS* como *Microsoft SQL Azure* [Azu, 2012] e *Google Megastore* [Baker et al., 2011].

Shared-Hardware: neste modelo, ilustrado na Figura 3.3, cada inquilino compartilha recursos de hardware, mas cada um obtém uma instância de banco de dados privada. Tal compartilhamento é alcançado com o emprego de MVs. Dessa forma, múltiplas MVs podem ser alocadas sobre o mesmo servidor. Cada MV geralmente corresponde a um inquilino e irá hospedar apenas um processo de banco de dados. Provendo assim, uma abstração ao banco de dados inquilino, como se o mesmo estivesse alocado em um hardware dedicado. Dessa maneira, o compartilhamento de recursos do SGBD não existe e as suas instâncias se mantêm independentes.

O modelo *share-hardware* não requer nenhuma modificação no banco de dados, sendo uma de suas principais vantagens. Na ocorrência de falhas de hardware ou no próprio SGBD, existe a facilidade de restaurar os dados de cada inquilino a partir de *backups*. Da mesma forma, a migração de servidores pode ser realizada facilmente, simplesmente movendo arquivos para o novo hospedeiro. Outra característica marcante deste modelo é oferecer um forte isolamento relacionado à segurança entre os bancos de dados inquilinos, uma vez que estão executando em MVs distintas.

Uma das desvantagens do modelo *shared-hardware* é que não há o compartilhamento de memória entre os bancos de dados inquilinos. Para cada SGBD é alocada uma grande quantidade de memória, simplesmente para a manipulação de uma instância do banco de dados. Dessa maneira, tal modelo não permite grande escalabilidade no que diz respeito

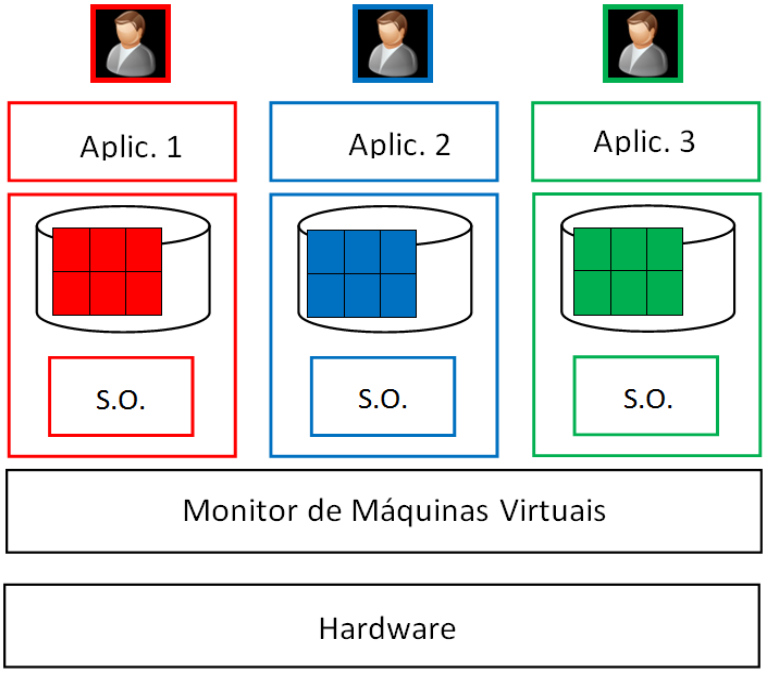


FIGURA 3.3: MODELO MULTI-INQUILINO *SHARED-HARDWARE*
FONTE: Modificado pelo autor [Agrawal et al., 2012]

à quantidade de clientes por servidor [Lazarov, 2007].

Outra desvantagem do modelo está relacionada à coordenação na utilização de recursos de hardware entre as MVs. Se tomarmos como exemplo o uso das unidades de disco, verificamos que o MMVs provê uma abstração de um disco virtualizado, o qual é compartilhado entre múltiplas MVs alocadas sobre o mesmo hospedeiro. Os SGBDs alocados sobre estas MVs, farão acesso não coordenado e concorrente ao disco, resultando em impacto considerável de desempenho. Assim, conforme o número de inquilinos que precisam ser consolidados sobre o mesmo servidor aumenta, o *overhead* sobre este modelo irá predominar [Agrawal et al., 2012].

O modelo *shared-hardware* é amplamente empregado quando é necessário que um pequeno número de inquilinos sejam consolidados sobre o mesmo servidor e também por provedores *IaaS*, como a *Amazon Web Services* (AWS) [Amazon Web Services, 2013].

3.2 Classificação dos Sistemas de Gerenciamento de Dados em Nuvem

São diversas as abordagens de SGBDs aplicáveis à computação em nuvem, cada uma com um conjunto de características próprias e propósitos específicos, o que agrava o problema de escolha por parte dos usuários. Assim, para facilitar o estudo, Souza *et al.* [Sousa et al., 2011], e Cattel *et al.* [Cattell, 2011] propõem a classificação dos SGBDs como *NoSQL* e *SQL/Relacionais*. Classificação esta que é discutida a seguir.

Sistemas NoSQL (um acrônimo para *Not Only-SQL*) é um termo genérico utilizado para representar uma ampla classe de SGBDs que não se utiliza do modelo relacional para a representação dos dados armazenados [Floratou et al., 2012, Stonebraker, 2010]. Projetados para atender a necessidade de armazenamento e gerenciamento de grandes volumes de dados semi-estruturados ou não-estruturados, podem ser classificados de acordo com o modelo que se utilizam para representação de dados, sendo os principais: Chave-valor, orientado a colunas, orientado a documentos, orientado a grafos e XML (*eXtensible Markup Language*) [Pokorny, 2011].

Os sistemas NoSQL apresentam características fundamentais que os diferenciam dos SGBDRs tradicionais como:

- **Escalabilidade Horizontal**: traduz a capacidade de distribuir os dados e a carga de operações entre diversos servidores sem haver o compartilhamento de memória principal ou disco entre estes. Para obter melhorias de escalabilidade e performance, os sistemas NoSQL apresentam um relaxamento nas características comuns de SGBDRs. Um exemplo, é a adoção parcial das propriedades transacionais ACID (*Atomicity, Consistency, Isolation, and Durability*) [Pritchett, 2008], onde operações de atualização são eventualmente propagadas, havendo garantias limitadas de consistência sobre operações de leitura.

- **Replicação e distribuição de dados**: possibilita a distribuição e também a replicação dos dados entre diversos servidores por meio de partições. Isto permite

suportar um maior número de operações simples de leitura/escrita por segundo. Tais operações são bastante comuns em aplicações *web* modernas.

- **Adição dinâmica de novos atributos**: apresenta maior flexibilidade do esquema do banco de dados, permitindo conforme a necessidade das aplicações, adicionar dinamicamente novos atributos para as tuplas de dados.

Sistemas SQL/Relacionais: Refere-se a classe SGBDs que se utiliza do modelo relacional, proposto pelo matemático britânico Edgar Frank Codd [Codd, 1970] como forma de sua representação e estruturação. Este modelo faz uso de tabelas nas quais os dados ficam armazenados empregando operações matemáticas para recuperá-los. A simplicidade proposta pelo modelo relacional que apresenta como característica básica a distinção entre aspectos físicos e lógicos de um banco de dados e a possibilidade de ocultar detalhes de implementação de seus usuários, mostra-se realmente interessante.

Nos anos subsequentes à definição do modelo relacional, começaram a surgir os primeiros protótipos de SGBDRs. Juntamente, foram propostas linguagens de alto nível como forma de representação das consultas a eles submetidas. Como exemplo, podemos citar a SQL (*Structured Query Language*) [Astrahan and Chamberlin, 1975], classificada como linguagem não-procedural onde os usuários descrevem quais dados desejam, sem especificar sua forma de obtenção, porporcionando assim um ambiente de utilização altamente produtivo.

Tradicionalmente, os SGBDRs não conseguem alcançar o mesmo grau de escalabilidade dos sistemas NoSQL. Porém, há pesquisas na tentativa de solucionar este problema. Há cerca de oito anos foi lançado o MySQL Cluster [MySQL Cluster, 2012], uma versão mais escalável, embora com desempenho mais baixo por nó, em relação ao MySQL [MySql, 2012] tradicional. Além do MySQL Cluster, existem outras soluções como o VoltDB [VoltDB, 2012], o Clustrix [Clustrix, 2012] e o Microsoft Azure [Azu, 2012] que prometem obter bom desempenho por nó e apresentam escalabilidade semelhante aos NoSQL porém com algumas restrições [Cattell, 2011]: (1) *Desenvolvimento de Pequenas*

Operações: como exemplo, pode-se citar a resolução de operações de junção sobre muitas relações, que envolvem diversos nós. Estas operações não apresentam boa escalabilidade. (2) *Desenvolvimento de Pequenas Transações:* transações que envolvem muitos nós tornam-se ineficientes devido ao *overhead* de comunicação gerado pelo protocolo *two-fase commit* [Silberschatz et al., 2010a, Ramakrishnan and Gehrke, 2008a].

A maturidade e a confiabilidade adquiridas pelos SGBDRs são o resultado de décadas de pesquisas, ajustes e aprimoramentos nesta arquitetura de armazenamento. A representação dos dados realizada de forma natural, independência, integridade e segurança em sua manipulação, além do emprego de linguagens de consultas com alto grau de abstração, tornam os SGBDRs ambientes altamente produtivos. Tais características fazem destes, componentes integrais e indispensáveis da grande maioria dos ambientes computacionais na atualidade.

O sucesso obtido pelos SGBDRs deve-se também em grande parte ao seu poder de adaptação às diferentes exigências do mercado feitas no decorrer dos anos. Como exemplo, as diferentes cargas de trabalho a ele impostas, a necessidade do processamento paralelo, o armazenamento de dados em memória principal e também distribuído em redes computacionais. Assim, é de se supor que SGBDRs ainda irão disputar espaço com novas tecnologias de armazenamento, porém, devendo agora se adaptar à novas exigências como a escalabilidade vertical (aumento e diminuição da capacidade computacional de cada nodo).

3.3 Bancos de Dados em Nuvem e os Sistemas Legados

Sistemas NoSQL emergiram como uma solução aos problemas de armazenamento de dados e são um tópico de discussão e pesquisas na atualidade. Atrelados à computação em nuvem, representam grandes avanços tecnológicos. Porém, mesmo a indústria de TI oferecendo taxas sem precedentes de inovações, existem casos em que é necessário manter em operação sistemas bastante antigos e até mesmo desatualizados. É comum a estes,

o encargo de manter adequadamente a regra de negócio das organizações. Tendo uma missão extremamente crítica, são chamados de *Sistemas de Informação Legados* (*Legacy Information Systems - LIS*) [Brodie and Stonebraker, 1995, Bennett, 1995]. Incorporá-los à infraestrutura de nuvem tem se tornado uma necessidade, já que resultam em altos investimentos a longo prazo. A tecnologia de virtualização é uma forma viável para atender a esse objetivo.

Muitos dos sistemas legados atualmente em operação foram projetados usando a tradicional arquitetura cliente/servidor. Tais sistemas são compostos principalmente por uma instância de um SGBDR e por um programa cliente que executa na estação de trabalho do usuário, o qual necessita ter acesso ao SGBDR por meio de uma rede local. Uma boa alternativa e que vem sendo amplamente empregada como forma de incorporar tais sistemas em nuvem é criar uma MV para hospedar a instância de SGBDR e criar outra, para hospedar a aplicação do cliente permitindo o acesso dos usuários aos dados armazenados.

A migração de SGBDRs legados para ambientes *shared-hardware*, pode se mostrar mais segura e flexível quando comparada aos outros modelos de implantação de sistemas de banco de dados multi-inquilinos já discutidos. Para isso, é necessária a criação de uma MV que atenda às necessidades do SGBDR, podendo ser idêntica em recursos à máquina física na qual o SGBDR estava anteriormente alocado. As rotinas relacionadas à migração do banco para o ambiente virtualizado, como *backup* do esquema e dos dados é facilitada, uma vez que são fornecidas pelo próprio SGBDR. Outra vantagem é que não são exigidas quaisquer alterações na aplicação de banco de dados como migração de versões ou adaptações em seu código-fonte.

Embora a virtualização tenha trazido grandes benefícios administrativos e econômicos para a manutenção de sistemas legados, sua flexibilidade no provisionamento de recursos tem potencializado o problema do ajuste de desempenho de SGBDRs, assunto este, que será discutido no Capítulo 4.

CAPÍTULO 4

OTIMIZAÇÃO DE DESEMPENHO DE SGBDR

Desde a sua concepção no início dos anos 70, os SGBDRs foram alvos de estudos com o objetivo de maximizar o seu desempenho e minimizar o consumo de recursos, caracterizando-se assim como um ou mais problemas de otimização. Devido à elevada opção de possíveis configurações, o processo de otimização desses sistemas é considerado de alta complexidade.

Um dos métodos de otimização de desempenho de SGBDRs amplamente estudado por mais de 30 anos está relacionado ao processamento de consultas, mais especificamente à fase de planejamento/otimização [Bini et al., 2011, Bini et al., 2009]. Outro método largamente empregado, refere-se ao *tuning* dos parâmetros de configuração do SGBDR. Assim, este Capítulo tem por objetivo apresentar e debater tais métodos de otimização, trazendo à discussão a inserção dos SGBDRs em ambientes de computação em nuvem, fazendo uso do modelo *shared-hardware* como forma de implementação para atender às requisições de sistemas legados.

4.1 Processamento de Consultas - Visão Geral

Os SGBDRs permitem serem fornecidas como entradas, consultas escritas em linguagens declarativas e não-procedurais, como por exemplo a SQL ou a OQL (*Object Query Language*) [Alashqur et al., 1989]. Tendo essas linguagens alto grau de abstração, não há a necessidade do usuário ou da aplicação se preocupar em como os dados armazenados serão obtidos para compor o resultado solicitado. Assim, é exigido dos SGBDRs, um mecanismo complexo para o processamento de consultas. Internamente tais sistemas implementam um conjunto de métodos para a manipulação de dados, podendo ser derivados da álgebra relacional [Codd, 1972]. Esses métodos são combinados e representados por

meio de um *plano de execução*, ou *plano físico* [1].

Uma consulta descrita em linguagens de alto nível pode ser representada por um grande número de planos de execução distintos. Mesmo cada plano sendo equivalente (gerando o mesmo resultado para o solicitante), os recursos computacionais e o tempo envolvido para processar cada um deles podem variar de forma considerável. A tarefa de escolher um plano físico para execução eficiente de determinada consulta é bastante crítica, sendo chamada de *planejamento*.

As diversas etapas envolvidas desde o momento do recebimento da consulta em linguagem de alto nível até a exibição dos resultados solicitados compõem o *processamento de consultas*. Este processo é apresentado na Figura 4.1 e detalhado nas próximas subseções.

4.1.1 Análise (Parsing)

O objetivo da etapa de análise é converter a consulta em uma árvore de análise cujos nós correspondem a: (1) *Átomos* que são elementos léxicos, como nomes de atributos, relações, constantes, palavras-chave (por exemplo, "SELECT", "FROM", "WHERE") e operadores (por exemplo, +, <, =). (2) *Categorias Sintáticas* que são nomes de sub-partes da consulta que desempenham papel semelhante, podendo ser representadas por um sinal de menor e maior (< >) ao redor de um nome descritivo [Garcia-Molina et al., 2008]. Por exemplo, <SFW> representa qualquer consulta com a forma *select-from-where* e <Condição> representando condições na cláusula "WHERE" em SQL.

Também é incumbência do analisador a análise sintática da instrução recebida, efetuada em um processo de três (3) etapas básicas [Garcia-Molina et al., 2008]:

- **Verificação de Uso das Relações:** verifica no esquema do banco de dados a existência de relações ou visões que foram mencionadas na cláusula "FROM" do SQL.

[1] *query evaluation plan* (QEP) [Swami and Gupta, 1988], *physical plan* [Garcia-Molina et al., 2008] ou *access plan* [Ioannidis, 1996]

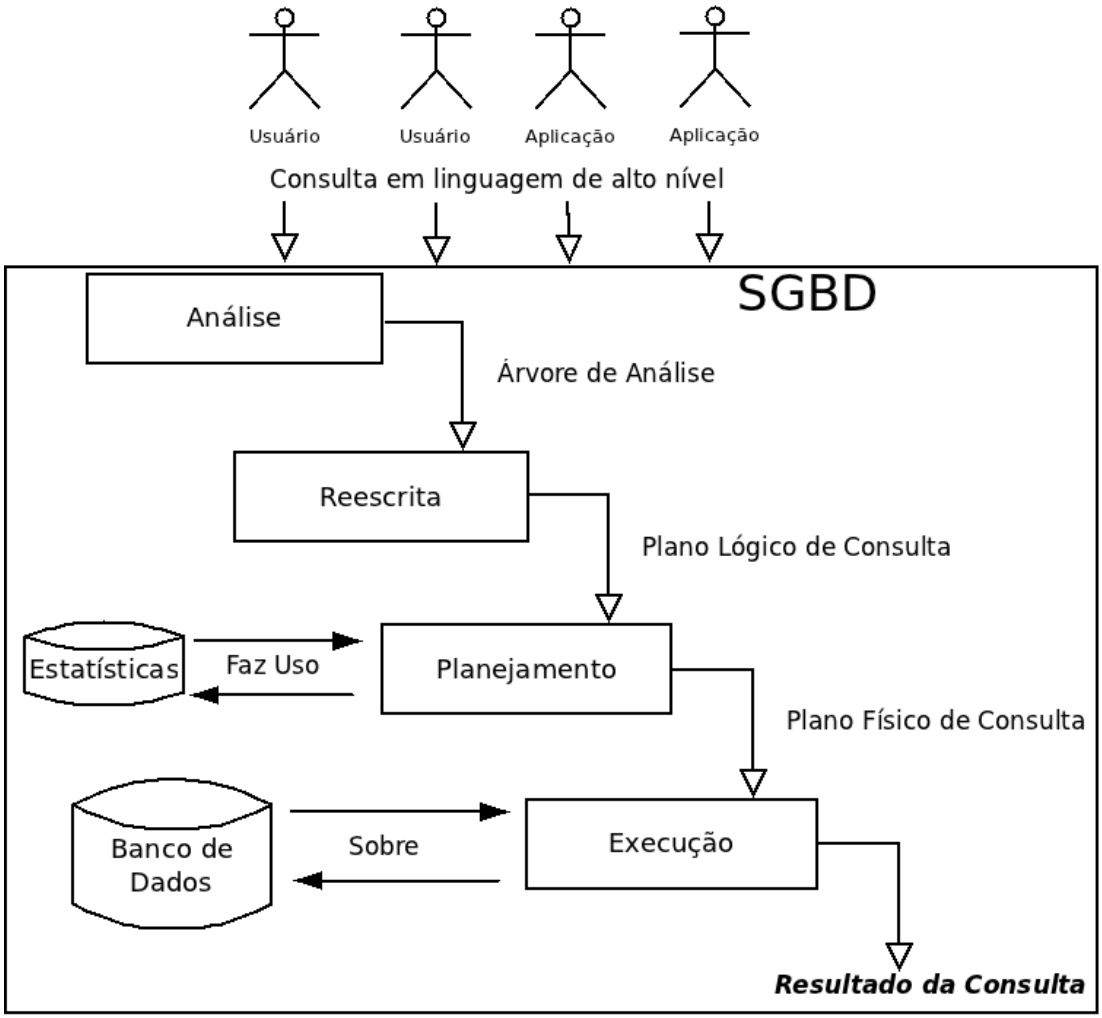


FIGURA 4.1: ETAPAS DO PROCESSAMENTO DE CONSULTAS EM UM SGBDR

- **Análise do Uso de Atributos:** tomando por base o esquema do banco de dados verifica se os atributos referenciados na cláusula "SELECT" ou "WHERE" da consulta SQL estão presentes nas relações referenciadas na cláusula "FROM". Em caso negativo, o analisador retorna mensagem de erro condizente.

- **Verificação de Tipos:** esta etapa analisa se os atributos têm seus tipos adequados quanto à sua utilização. Por exemplo, a possibilidade de efetuar a comparação de um atributo do tipo *string* com outro atributo do tipo *data*. De forma semelhante, os operadores são verificados confirmando sua aplicação.

Caso todas as condições sejam atendidas, o resultado deste processo pode ser representado

por meio de uma árvore, chamada de *árvore de análise* (*parse tree*) ou outra representação que descreva a forma declarativa da consulta. Caso as condições não sejam válidas, uma mensagem apropriada será emitida e nenhum processamento adicional irá ocorrer.

4.1.2 Reescrita (Rewrite)

Na etapa de reescrita, a árvore de análise é convertida em um plano de consulta inicial, uma representação algébrica da consulta. Logo após, o mesmo é transformado em um plano equivalente, o qual deverá exigir menor tempo e recursos (processamento, memória, acessos a disco) para sua execução. Este plano é chamado *plano lógico de consulta* e para ser obtido, são necessárias duas (2) etapas básicas [Garcia-Molina et al., 2008]: (1) Substituição dos nós e estruturas da árvore de análise por um ou mais operadores da álgebra relacional. (2) A partir da expressão algébrica produzida na etapa anterior, transformá-la em uma expressão que será convertida em um plano físico de consulta mais eficiente.

Considerando determinada consulta SQL, a mesma pode ser traduzida em várias expressões equivalentes em resultado. Baseado em leis algébricas [Garcia-Molina et al., 2008, Ramakrishnan and Gehrke, 2008b] válidas para a álgebra relacional, variadas expressões são analisadas buscando a simplificação do ponto de vista algébrico. A de maior eficiência é escolhida, sendo conhecida como plano lógico da consulta. Neste contexto são exemplos, as leis associativas e comutativas, leis que envolvem seleções, projeções e produtos, a eliminação de tuplas duplicadas das relações, além de alterações na ordem de execução das seleções e projeções ("empurar" seleções e projeções), entre outras. É importante ressaltar que nem todas as possíveis expressões algébricas são analisadas, devido a sua grande quantidade. Assim, o otimizador pode concentrar-se em partes do espaço de busca, diminuindo assim o esforço computacional para a geração do plano lógico.

4.1.3 Planejamento

Depois da consulta ter sido devidamente analisada e transformada em um plano lógico de consulta, este por sua vez, deve ser transformado em um *plano físico*. Este indica não somente as operações a serem executadas, mas também a sua ordem, assim como os algoritmos que devem ser empregados neste processo, a forma de obtenção dos dados armazenados e o repasse destes entre as diversas operações envolvidas.

Como mencionado, existem diversas expressões algébricas correspondentes à mesma consulta SQL. Da mesma maneira existem diversos planos físicos, para uma mesma expressão algébrica. A Figura 4.2 exemplifica de forma genérica, por meio de um modelo entidade-relacionamento, a equivalência de cada representação utilizada por um SGBDR.

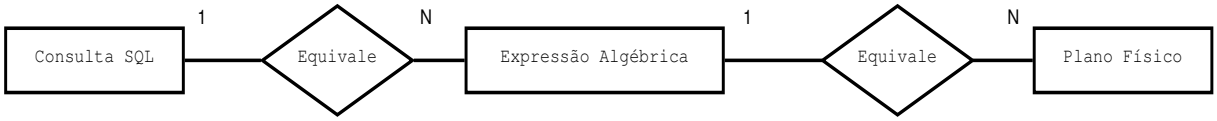


FIGURA 4.2: EQUIVALÊNCIA ENTRE UMA CONSULTA SQL E SEUS RESPECTIVOS PLANOS FÍSICOS

FONTE: [Lange, 2010]

Uma vez que são analisados diversos planos físicos, o custo de cada plano é avaliado individualmente levando em consideração [Garcia-Molina et al., 2008]:

1) A ordem das operações associativas e comutativas como junção, união e interseção.

2) O método de junção utilizado para o processamento de junções ou produtos cartesianos. Podem ser baseados em *loops* aninhados (*nested loop join*), em ordenação (*merge join* ou *sort merge join*) ou baseados em tabelas *hash* (*hash join*). Tais métodos podem sofrer variações dependendo da arquitetura de armazenamento e processamento do SGBD [Lange, 2010].

3) O método de acesso que define a forma de acesso aos dados contidos nas relações base da consulta. Podem ser classificados em sequenciais (*seq-scan*) ou em índices (*index-scan*). Ambos os métodos dependem de características físicas de armazenamento do SGBDR.

4) A forma de repasse dos argumentos entre os operadores. O resultado pode ser armazenado temporariamente em disco (materialização), ou fazer uso de iteradores, repas-

sando um argumento de um *buffer* em memória a cada vez [Garcia-Molina et al., 2008].

4.1.3.1 Estimando Custos para Planos de Execução

É necessária a análise das diversas questões mencionadas para a geração e posteriormente a escolha do plano físico de maior eficiência. Tal eficiência reflete na menor utilização de recursos computacionais e na execução de forma rápida do plano físico. Porém, não podemos conhecer o custo de cada uma das operações descritas, tão pouco o custo total de cada plano físico de consulta gerado sem a execução dos mesmos. Sem dúvida, tal tarefa torna-se inviável, sendo o SGBDR obrigado a estimar o custo de cada plano sem executá-lo. Assim, as estimativas utilizadas pelos SGBDRs para tamanhos e custos dos resultados são apenas aproximações dos dados reais. Dessa forma, dificilmente o otimizador irá encontrar o melhor plano físico para uma consulta. Nesta etapa o importante é evitar os piores planos e encontrar um bom plano (plano subótimo) [Ramakrishnan and Gehrke, 2008c].

O custo computacional estimado para um plano físico de uma consulta pode ser medido tendo como base vários recursos distintos que incluem entre outros o acesso a disco e o tempo de processamento. Em grandes sistemas de banco de dados o custo para o acesso aos dados armazenados em disco é o mais importante, visto a sua lentidão comparada às operações em memória principal. Além disso, se verifica que as unidades de processamento apresentam velocidades muito superiores em relação aos discos. Neste contexto, se pressupõem que o tempo envolvido com operações de disco continue a dominar o tempo total de execução das consultas. Por fim, em aplicações reais o custo de processamento empregado na execução de determinada tarefa é difícil de estimar, dependendo de detalhes de baixo nível do código em execução [Silberschatz et al., 2010b]. Baseado nestes argumentos, a literatura [Ramakrishnan and Gehrke, 2008c, Silberschatz et al., 2010b] considera apenas os custos de acesso a disco para medir os custos de um plano físico de execução de consulta.

O módulo que realiza estimativas de custos é um dos componentes mais críticos dos

otimizadores relacionais. É fundamental a estes contar com procedimentos precisos para a estimativa de custos uma vez que são invocados repetidas vezes durante o processo de otimização [Bruno, 2003]. Um framework básico para a estimativa de custo baseia-se na seguinte abordagem recursiva [Chaudhuri, 1998]:

1) Coleta de resumos estatísticos dos dados armazenados: Os otimizadores de consultas fazem uso de informações estatísticas armazenadas no catálogo do SGBDR (podem ser armazenadas na forma de relações) para estimar o custo dos planos. Entre as diversas informações relevantes à relações podemos citar:

n_R: número de tuplas da relação R.

s_R: tamanho em bytes de uma tupla da relação R.

$s_R(a_i)$: tamanho (em bytes) do atributo a_i da relação R.

f_R: fator de bloco da relação R, ou seja, número de tuplas da relação R que cada bloco é capaz de suportar. Dado por: $f_R = t_{bloco} \ / \ t_R$.

b_R: número de blocos necessários para manter as tuplas da relação R.

$V_R(a_i)$: quantidade de valores distintos para o atributo a_i considerando a relação R.

$C_R(a_i)$: cardinalidade (estimada) do atributo a_i da relação R (tuplas da relação R que satisfazem um predicado de igualdade sobre a_i.

$GS_R(a_i)$: grau de seletividade do atributo a_i da relação R.

É bastante comum a literatura utilizar o termo *página* como um *bloco* do banco de dados armazenado em memória principal. A real distinção entre estes dois termos empregados alternadamente em muitos textos é que uma página em memória é capaz de manter um bloco, juntamente com uma pequena quantidade de informações. Este *overhead* é usado para identificação do bloco sendo conhecido como *buffer header* [Smith, 2010]. Neste trabalho utilizaremos somente o termo bloco como unidade de armazenamento de dados e transferência dos mesmos do disco à memória principal.

É importante destacar que informações estatísticas sobre uma coluna ou um conjunto de colunas que compõem uma relação, se disponíveis, podem ajudar, por exemplo, a estimar a cardinalidade de predicados de junção. Neste contexto, os *histogra-*

mas [Piatetsky-Shapiro and Connell, 1984, Bruno, 2003] são técnicas amplamente empregadas por SGBDRs para representar tais estatísticas.

Além de informações sobre relações, é comum no catálogo dos SGBDRs serem encontrados também informações sobre índices, como:

f_i: *(fan-out)* fator de bloco do índice i (quantos nodos de uma árvore B^+ cabem em um bloco).

h_i: número de níveis (de blocos) do índice para valores de um atributo a_i ("altura" do índice).

bf_i: número de blocos de índice no nível mais baixo do índice (número de blocos "folha").

2) Estimar os custos de execução de um operador: O custo estimado para um plano é obtido também pela combinação de custos de cada um de seus operadores. Neste contexto, deve-se considerar as relações intermediárias, argumento dos operadores na expressão do plano lógico. A acurácia na estimativa de suas cardinalidades, a forma como as mesmas serão armazenadas (agrupadas ou não agrupadas, indexadas ou não indexadas) afetam drasticamente a estimativa de custos de um determinado plano de execução [Babcock and Chaudhuri, 2005].

É necessário que o SGBDR mantenha seu catálogo e suas estatísticas atualizadas. Porém, esta tarefa gera um *overhead* significativo principalmente em bancos onde são realizadas transações do tipo OLTP (*Online Transaction Processing*) [Harizopoulos et al., 2008]. Estas transações se caracterizam pela grande quantidade de inserções, atualizações e exclusões de tuplas. Assim, a maioria do SGBDRs não atualiza as estatísticas conforme as alterações na base são realizadas. Como resultado, as estratégias empregadas no processamento de consultas podem não ser precisas.

4.1.4 Execução

O executor de consultas tem por objetivo interpretar e executar o plano físico selecionado na etapa anterior. Cada método descrito pelo plano é então acionado em sua

ordem pré-determinada. Nesta fase também são requisitados recursos de hardware, como processamento, reserva de memória principal e requisições de leitura e escrita nos meios de armazenamento secundários. Como resultado, tem-se o retorno dos dados solicitados pela consulta.

4.2 *Tuning* de Sistemas Gerenciadores de Banco de Dados

Além da otimização de consultas, outro método para se otimizar o desempenho de SGBDR é o uso de técnicas de *tuning*. Segundo Shasha *et al.* [Shasha and Bonnet, 2002], o *tuning* de SGBDs diz respeito às atividades necessárias para permitir que o mesmo execute mais rapidamente, resultando em menor tempo de resposta às solicitações das aplicações. Para esse intuito, o responsável por estas atividades pode alterar as formas como as aplicações são desenvolvidas, assim como as estruturas de dados e parâmetros do SGBD, além da configuração do sistema operacional e do hardware envolvidos.

Geralmente, tratando-se de SGBDR, existe um grande número de parâmetros de configuração que podem ser ajustados, alterando significativamente seu desempenho. Estes parâmetros controlam, por exemplo, a distribuição de memória, as entradas do modelo de custo do otimizador de consultas, *logs* e outros aspectos. Assim, na tentativa de facilitar a compreensão e estudo, os parâmetros de configuração empregados em técnicas de *tuning* geralmente são classificados em dois (2) tipos [Soror et al., 2008, Soror et al., 2007]: (1) *Prescriptive parameters*: os quais controlam as configurações do SGBDR afetando-o diretamente. Como exemplo, pode-se citar parâmetros relacionados à quantidade de memória compartilhada pelo SGBDR. (2) *Descriptive parameters*: são parâmetros que afetam indiretamente o SGBDR, na estimativa de custos do otimizador de consultas. Neste contexto, determinado parâmetro que defina o custo para operações de acesso ao disco pelo SGBDR é um exemplo interessante.

Encontrar boas configurações para esses parâmetros por meio de técnicas de *tuning* é uma tarefa desafiadora e que demanda tempo, devido às complexas formas com que a con-

figuração dos parâmetros podem afetar o desempenho do sistema [Sullivan et al., 2004]. Métodos de tentativa e erros para obter boas definições de parâmetros são comumente empregados, tomando por base uma réplica do banco de dados de produção. Para análise de desempenho, são executados testes com diferentes definições de valores para os parâmetros, sob cargas de trabalho representativas. O processo é repetido até a obtenção de uma definição, que atinja os objetivos de desempenho. Isso se torna extremamente laboroso e requer entre outros, profundos conhecimentos sobre o SGBDR, plataforma operacional e recursos de hardware envolvidos.

Tradicionalmente as atividades relacionadas ao *tuning* de SGBDs são realizadas manualmente por especialistas, como administradores de banco de dados, porém, isto tem se revelado cada vez mais inviável. Com a grande queda nos preços do hardware e o aumento na capacidade de gestão de dados, os sistemas de banco de dados tem se tornado gradativamente maiores e mais complexos. Além disso, as cargas de trabalho submetidas aos bancos cada vez mais se caracterizam como heterogêneas e dinâmicas. Assim, a mão de obra especializada exigida pelas atividades de *tuning* é extremamente escassa e onerosa às organizações.

Atualmente é bastante comum aos desenvolvedores de grandes SGBDR o oferecimento de ferramentas de **auto-configuração** (*self-tuning*). Seu objetivo é realizar o ajuste e configuração de desempenho de suas aplicações de forma automática e dinâmica. Ferramentas como o *Database Diagnostic Monitoring* [Dias et al., 2005] da Oracle, o *Resource Advisor* [Narayanan et al., 2005] para o SQL Server e o *Self-Tuning Memory Manager* [Storm et al., 2006] para o DB2 merecem ser destacadas. Outro exemplo interessante, não comercial e aplicável ao SGBDR PostgreSQL é a ferramenta *Ituned* [Duan et al., 2009], que analisa de forma *on-line* bases de dados de produção na tentativa de encontrar, através de métodos de amostragem, boas definições para parâmetros de configuração do SGBDR. Parâmetros que causam maior impacto na melhoria de desempenho são escolhidos, procurando causar o mínimo possível de *overhead* à carga de trabalho de produção.

4.3 Otimização de Desempenho de SGBDR em Nuvem

Antes do surgimento da computação em nuvem, uma prática bastante comum era super-dimensionar o hardware disponibilizado ao SGBDR de forma a suportar o pico da carga de trabalho a ele aplicado, mesmo que este representasse uma pequena fração do tempo total da utilização do sistema. Uma vez que o hardware eram super-dimensionados, o SGBDR operava com sobra de recursos em boa parte de seu funcionamento, diminuindo a possibilidade de apresentar um desempenho não satisfatório. Além disso, tal super-dimensionamento reduzia a necessidade de reconfigurações no SGBDR exceto em momentos onde a carga de trabalho fosse mais intensa.

Considerando-se ambientes de computação em nuvem, tanto a subutilização de recursos computacionais como o super-dimensionamento de hardware são práticas indesejáveis, uma vez que aumentam de forma significativa (no contexto de larga escala, com centenas ou mesmo milhares de máquinas), os custos de infraestrutura repassados aos clientes. Uma solução é o compartilhamento de recursos entre os vários clientes, principalmente se os picos de utilização de seus serviços não coincidirem. Uma das formas de prover o compartilhamento de recursos considerando SGBDR é a sua implantação sobre o modelo multi-inquilino do tipo *shared-hardware*, discutido no Capítulo 3.

De um modo geral, existe um conflito importante entre maximizar o desempenho de um SGBDR e minimizar os recursos por ele utilizados. Uma vez que estes não foram inicialmente projetados para serem executados em ambientes *shared-hardware*, seus parâmetros de configuração não consideram o fato de que os recursos disponíveis possam variar ao longo do tempo. Tão pouco o modelo de custos dos atuais SGBDRs, base para a tomada de decisões e otimizações, leva em consideração sua execução em um ambiente elástico, que implica em um provisionamento dinâmico de recursos. Neste contexto, a existência de cargas de trabalho concorrentes, oriundas de outras MVs sobre um mesmo hardware, é um exemplo que também não pode ser ignorado. Forçar o isolamento de desempenho entre MVs, por meio de ajustes nos escalonadores do monitor de máquinas virtuais, pode não

ser completamente possível em alguns casos, além de não ser economicamente desejável.

Para que o ajuste de desempenho do SGBDR em ambiente *shared-hardware* seja eficiente, deve-se considerar a existência de cargas de trabalho de outras MVs, alocadas sobre o mesmo servidor e que concorrem por recursos físicos. A necessidade de constantes acessos às unidades de disco, comuns em ambientes de produção, representam o maior "gargalo" para SGBDRs que manipulam grandes quantidades de dados [Hsu et al., 2001]. Além disso, é reconhecido por alguns autores, que o acesso a disco é um recurso de difícil isolamento em ambientes virtualizados [Gulati et al., 2010, Gupta et al., 2006]. Considerando as unidades de disco como dispositivos mecânicos, cargas de trabalho de disco podem ser caracterizadas basicamente em dois (2) tipos: Em uma dimensão requisições de *Leitura* e *Escrita*. Em outra dimensão, acessos *Aleatórios* e *Sequenciais* ao disco [Delimitrou et al., 2012].

Para o otimizador de consultas dos SGBDR operar de forma satisfatória em ambientes *shared-hardware* seus mecanismos de auto-configuração devem ser capazes de reconhecer o dinamismo do provisionamento de recursos gerado por tal ambiente e se adaptar a ele. Uma vez que as informações sobre este provisionamento não são de domínio do SGBDR e sim do MMVs, responsável pelo escalonamento de recursos, é necessária a concepção de uma nova arquitetura de custos que deve definir as informações trocadas entre o MMVs e o SGBDR. Isso permitirá a manutenção de seu modelo de custos e uma maior precisão nas suas estimativa, resultando em planos de execução de consultas mais eficientes e em um melhor desempenho.

O desenvolvimento de uma nova arquitetura de custos, mesmo que restrita a operações de acesso a disco resultará em reescrita do código do SGBDR. Está não é uma solução viável e desejável quando considerada sua utilização por sistemas legados. Estes sistemas, muitas vezes são dependentes de versões estritas do SGBDR comumente descontinuadas e obsoletas. Nestes casos, uma solução menos invasiva é o emprego de técnicas de *tuning* como forma de otimizar o desempenho do SGBDR. Para esta finalidade, regras de ajuste de parâmetros de configuração geralmente encontradas na literatura e/ou recomendada

por *experts* (Regras-de-Ouro / *Rules of Thumb*) podem ser utilizadas.

No contexto descrito, torna-se interessante e fundamental a apresentação de um método para analisar o comportamento dos SGBDRs considerando principalmente seu desempenho quando inseridos em ambientes virtualizados. Mais especificamente, ambientes que tomam por base o modelo *shared-hardware*, comumente e amplamente empregado como forma de redução de custos operacionais pelas organizações. Este método deverá simular a presença de cargas de trabalhos que realizam manipulação de dados armazenados em disco, de forma concorrente à execução do SGBDR, pois são oriundas de outras MVs alocadas sobre o mesmo servidor. Tais cargas de trabalho devem ser devidamente caracterizadas pelo seu tipo e pela forma de acesso às unidades de disco.

O Capítulo 5 apresenta e detalha características de implementação deste método, o qual é capaz de analisar e contestar a adequação das regras-de-ouro aplicadas a SGBDRs inseridos em ambientes virtualizados. Por meio deste método, diferentes regras de configuração podem ser experimentadas frente a concorrentes cargas de trabalho de disco, caracterizadas nas duas (2) dimensões citadas. Isto possibilita a definição de novas regras-de-ouro para a configuração de SGBDRs em ambientes virtualizados e a possibilidade de desenvolvimento de inúmeras pesquisas relacionadas.

CAPÍTULO 5

ROTEIRO DE AVALIAÇÃO EXPERIMENTAL

Para realização dos experimentos foi utilizado um computador com processador Intel Core I7 (975 *Processor Extreme Edition*) de 3.33 GHz, com 8 MB de memória *cache* L2 e um total de 24 GB de memória RAM (*Random Access Memory*). Como memória secundária, foram empregados 4 discos rígidos SATA de 2 TB e 7200 RPM cada, dispostos em RAID (*Redundant Array of Independent Disks*) 10. O sistema operacional utilizado no hospedeiro foi o GNU/Linux com kernel 3.13.11X86-64. Foi aplicado o qemu-kvm [QEMU-KVM, 2013] versão 2.0.0 para a implementação do ambiente virtualizado. Cada uma das oito (8) MVs criadas e gerenciadas pelo ambiente gráfico Virtual Machine Manager versão 0.9.1 [Virt Manager, 2014] executavam o GNU/Linux, kernel 3.14.12X86-64. Como SGBDR foi empregado o PostgreSQL [PostgreSQL, 2013] versão 9.3.0, devidamente instalado sobre uma (1) das oito (8) MVs criadas. A quantidade de máquinas virtuais foi definida tomando como base o número de CPUs virtuais/lógicas disponíveis. Para todas as MVs utilizadas nos experimentos foram dados os mesmos recursos: uma CPU virtual, 512 MB de memória RAM e 48 GB de espaço em disco.

5.1 Parâmetros de Configuração do SGBDR

Um SGBDR típico apresenta diversos parâmetros de configuração, conforme discutido e o PostgreSQL não é uma exceção à regra. Assim, definir valores para os mesmos de forma adequada resultando em desempenho aceitável é uma tarefa bastante crítica. Avaliar o efeito de todos os possíveis valores em cada parâmetro de configuração exige um número exponencial de experimentos. Para evitar tal problema, foram considerados os trabalhos de [Debnath et al., 2008b, Debnath et al., 2008a]. Estes autores realizaram uma classificação (*ranking*) dos parâmetros do SGBDR PostgreSQL que exercem maior

influência no processo de otimização de desempenho através de regras de *tuning*. Baseado neste estudo, três (3) dos parâmetros mais bem colocados em sua classificação foram adotados. Tais parâmetros são do tipo *prescriptive*, os quais podem ser alterados ou visualizados a partir do arquivo de configuração *postgresql.conf* do PostgreSQL. Estes parâmetros serão detalhados a seguir:

1) Shared_buffers: parâmetro relacionado à memória compartilhada do PostgreSQL, que controla o tamanho do bloco em memória destinado ao armazenamento de dados a serem gravados ou já lidos pelo banco de dados. Dessa forma, o PostgreSQL não realiza operações de leitura e escrita de dados diretamente no disco, mas utiliza primeiramente a memória *cache* compartilhada. Caso não encontre tais dados, uma requisição ao sistema de arquivos é realizada, para acesso ao disco. Isto permite que o SGBD aproveite melhor o seu *cache* e faça um número menor de requisições ao disco rígido, melhorando seu desempenho. O valor padrão adotado na versão 9.3.0 do PostgreSQL para o parâmetro *shared_buffers* é 128 MB.

2) Effective_cache_size: Define a quantidade de memória RAM que será utilizada para o *cache* efetivo do banco de dados, sendo definido como o total de *shared_buffers* mais o tamanho do *cache* do *buffer* de disco do sistema operacional, após a base de dados ser iniciada. Isto acaba por ser maior do que a metade da memória total do sistema considerando um servidor de banco de dados dedicado típico. Essa configuração, na prática, faz com que o SGBDR não precise de constantes leituras de tabelas e índices a partir do disco, mantendo-os em memória, em virtude do acesso ao disco ser mais custoso. Esse parâmetro não aloca qualquer memória em si, simplesmente serve como um valor de consultoria para o planejador do SGBDR sobre o que provavelmente deve estar disponível de memória. Caso seu valor seja reduzido, menor será a probabilidade de escolha de um plano de execução de consulta eficaz pelo SGBDR. O valor padrão adotado para o parâmetro *effective_cache_size* é de 128 MB.

3) Work_mem: Tal parâmetro serve como limitador da quantidade de memória disponibilizada para operações de classificação e ordenação do SGBDR, ou seja, este

parâmetro coloca um "teto"na quantidade máxima de memória que uma única operação ocupa de RAM antes de ser enviada de forma forçada para o disco. Este parâmetro é importante em ambientes do tipo OLAP (*Online Analytical Processing*) e DSS (*Decision Support System*) caracterizado por operações de ordenação bastante complexas. Para sua devida configuração, deve-se levar em consideração o parâmetro *max_connections* que tem seu valor padrão disposto pelo desenvolvedor como cem (100) e não sofreu qualquer alteração durante a realização dos experimentos. O valor padrão adotado pelo PostgreSQL para o parâmetro *work_mem* é 1 MB.

A Tabela 5.1 apresenta os três (3) parâmetros de configuração do PostgreSQL assim como seus valores máximos e mínimos. Tais intervalos seguem as recomendações que constam na documentação do próprio SGBDR [PostgreSQL, 2013], assim como referências encontradas na literatura especializada [Smith, 2010].

TABELA 5.1: PARÂMETROS DE CONFIGURAÇÃO DO POSTGRESQL UTILIZADOS NOS EXPERIMENTOS E VALORES SUGERIDOS PARA SUAS CONFIGURAÇÕES

Parâmetro	**Valor Mínimo**	**Valor Máximo**
shared_buffers	25 % do total de memória RAM	50 % do total de memória RAM.
effective_cache_size	50 % do total de memória RAM.	75 % do total de memória RAM.
work_mem	Total de RAM / *max_connections* / 16	Total de RAM / *max_connections* / 4

FONTE: [Elmore et al., 2011]

Os experimentos utilizaram os seguintes valores e porcentagens sobre a memória RAM para os parâmetros de configuração do PostgreSQL:

- **shared_buffers:** 2,5% - 5% - 25% - 40% - 70% - 150%, sendo 25% o valor padrão adotado pelo SGBDR e 40% adotado como o recomendado para nossa configuração de hardware (regra-de-ouro). O valor configurado como 150% é uma recomendação encontrada em fóruns e discussões sobre *tuning* do PostgreSQL e não na documentação do PostgreSQL ou literatura especializada.

- **effective_cache_size:** 10% - 25% - 40% - 60% - 90%, sendo 25% o valor padrão adotado pelo SGBDR e 60% adotado como o recomendado para nossa configuração de hardware (regra-de-ouro).

- **work_mem:** 300 KB - 1 MB - 3 MB, sendo 1 MB o valor adotado como o recomendado para nossa configuração de hardware (regra-de-ouro).

Finalizamos nossa discussão a respeito dos três (3) parâmetros de configuração analisados nos experimentos apresentando a Tabela 5.2 que ilustra seus "equivalentes" considerando os SGBDRs MySQL [MySql, 2012] e Oracle [Oracle, 2013]. Como é possível verificar, esses sistemas possuem parâmetros com características similares ao do PostgreSQL. Porém, a equivalência entre os parâmetros de seus respectivos SGBDRs não pode ser realizada diretamente tão pouco a de seus referidos valores.

TABELA 5.2: COMPARATIVO DOS PARÂMETROS DE CONFIGURAÇÃO DO POSTGRESQL, MYSQL E ORACLE

PostgreSQL	**MySQL**	**Oracle**
shared_buffers	*query_cache_size*	*db_cache_size*
work_mem	*join_cache_size* *sort_cache_size*	*sort_area_size* *join_area_size* *key_area_size*
effective_cache_size	*key_cache_size*	*optimizer_index_cost_adj* *key_area_size*

5.2 Carga de Trabalho do Banco de Dados

Como forma de permitir a realização dos mesmos experimentos por outros pesquisadores, fez-se uso da base de dados sintética provida pelo *benchmark* TPC-H [TpcApp, 2012]. Seu esquema é composto por um total de oito (8) relações divididas em: (1) *tabela de dimensões* que armazenam dados que descrevem os elementos do negócio e (2) *tabelas de fatos*, as quais armazenam medições a respeito do negócio além de conter chaves para as tabelas de dimensões. Foi configurado com dez (10 GB) o *Fator de Escala* (*SF - Scale Factor*), componente que define o tamanho da base de dados gerada. Com índices, o

tamanho da base de dados experimental foi de dezessete (17) GB. A Figura 5.1 compreende a estrutura do banco de dados criada pelo *benchmark* TPC-H, apresentando suas tabelas, o relacionamento entre seus atributos e os dados numéricos que serão multiplicados pelo *Fator de Escala*, definindo assim a cardinalidade de cada tabela.

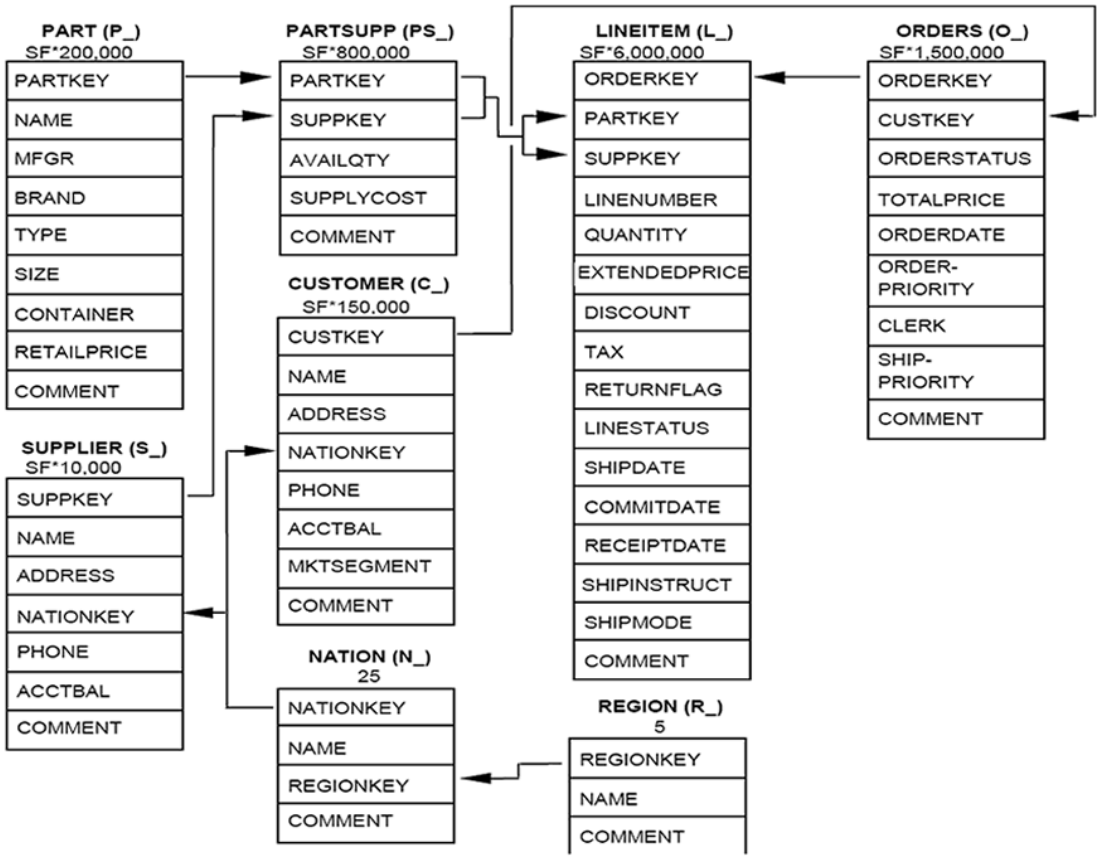


FIGURA 5.1: REPRESENTAÇÃO DO BANCO DE DADOS DO *BENCHMARK* TPC-H

FONTE: [TpcApp, 2012]

A carga de trabalho proporcionada por este *benchmark* é composta por um total de vinte e duas (22) consultas de alta complexidade e com capacidade de acesso à grandes quantidades de dados armazenados no banco de dados, além de elevadas exigências de processamento. Isto permite a implementação de um ambiente analítico (OLAP) e de extração de informações (DSS) que mede a capacidade de processamento e o volume de dados suportado pelos SGBDRs.

A escolha do *benchmark* TPC-H é justificada uma vez que nossos estudos se concen-

tram principalmente nos efeitos causados ao SGBDR e em suas configurações de *tuning* pelo acesso concorrente a disco entre as MVs. Como sua carga de trabalho é caracterizada além do complexo acesso a disco, pelas constantes requisições de processamento, algumas alterações foram necessárias para realização dos experimentos. Assim, foi gerado um conjunto de dezoito (18) consultas SQL que são derivadas das originais providas pelo *benchmark*. Tais consultas apresentadas no Apêndice A, não realizam operações de junção, ou seja, há maior concentração em operações de acesso a disco, ocasionando a não utilização de todas as consultas providas pelo TPC-H. As consultas resultantes seguem a nomenclatura deste *benchmark*, porém as que sofreram alterações receberam nomes padronizados da seguinte forma: *"nome_da_consulta_original.versão.sql"*. Por exemplo, a consulta *16.sql* originalmente fornecida pelo TPC-H foi adaptada, resultando em duas (2) novas consultas: *16.1.sql* e *16.2.sql*.

A interação com a base de dados fornecida pelo *benchmark* TPC-H, foi realizada por meio da ferramenta *psql* provida pelo PostgreSQL, que permitiu a submissão das consultas SQL e a visualização de seus resultados. Foi habilitado o comando */timing* do *psql* que demonstra o tempo em milissegundos (ms) para que a consulta apresente seus resultados ao solicitante. Neste intervalo de tempo, são contabilizados, entre outros, o tempo para geração dos planos de execução e o *delay* da rede.

Informações detalhadas sobre o plano de execução da instrução SQL como o custo estimado para cada operação a ser executada, além do custo total obtido pelo plano escolhido são obtidas pelos comandos *EXPLAIN* e *ANALYZE* no *psql* (*EXPLAIN ANALYZE + Consulta SQL*). Como principais informações reveladas pelo comando *EXPLAIN* [PostgreSQL, 2013] têm-se:

- **Startup Cost:** corresponde a um custo estimado de um nó (operação no plano), até o momento imediatamente anterior ao retorno na primeira tupla;
- **Total Cost:** corresponde a estimativa apresentada por um nó para o retorno de todas as suas tuplas. Este valor é afetado se especificado a cláusula *LIMIT* que

restringe a quantidade de linhas a serem apresentadas;

- **ROWS:** estimativa de linhas retornadas para determinado nó do plano;
- **WIDTH:** estimativa da quantidade de *bytes* retornados para determinado nó do plano.

Quando utilizado somente o comando *EXPLAIN* a consulta não é realmente executada. São apenas informados dados referentes ao plano de execução escolhido. A opção *ANALYZE* por sua vez, faz com que a consulta seja executada, e não apenas planejada. O tempo total de duração de cada etapa do plano (*actual time* - em milissegundos) e o número total de linhas (*rows*) realmente retornadas, são adicionadas ao resultado. Esta opção é útil para analisar se as estimativas do planejador estão próximas da realidade. Um exemplo das informações apresentadas pelo comando *EXPLAIN ANALYZE* sobre uma consultas no PostgreSQL é iustrado na Figura 5.2.

```
                                                                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=7114.93..7115.18 rows=100 width=136) (actual time=1358.014..1358.100 rows=100 loops=1)
   ->  Sort  (cost=7114.93..7364.93 rows=100000 width=136) (actual time=1358.008..1358.052 rows=100 loops=1)
         Sort Key: s_acctbal, s_name
         Sort Method: top-N heapsort  Memory: 49kB
         ->  Seq Scan on supplier  (cost=0.00..3293.00 rows=100000 width=136) (actual time=22.774..1270.191 rows=100000 loops=1)
 Total runtime: 1406.563 ms
(6 rows)
```

FIGURA 5.2: EXEMPLO DE PLANO DE EXECUÇÃO DE CONSULTA NO POSTGRESQL

Durante a execução dos experimentos, cada consulta foi submetida por cinco (5) vezes ao SGBDR, considerando cada um dos três (3) parâmetros descritos (*shared_buffers*, *effective_cache_size*, *work_mem*), assim como seus respectivos intervalos de porcentagens e valores. Logo após, foi calculada a média das cinco (5) execuções. Este processo foi repetido para cada uma das quatro (4) cargas concorrentes de acesso a disco gerada pelas sete (7) MVs distintas a qual foi hospedado o SGBDR. Destaca-se que não foi realizada qualquer operação de limpeza de memória *cache* entre a execução de cada consulta (*cache* quente). Para todo este processo que resulta na coleta dos dados, ocorreram cerca de cinco mil e quatrocentas (5.400) submissões de consultas ao SGBDR, sendo que o tempo

gasto para realização dos experimentos foi de aproximadamente quatrocentas e oitenta (480) horas de testes.

5.3 Cargas de Trabalho de Acesso a Disco (concorrentes ao SGBDR)

Foi empregado o *benchmark* bonnie++ [Bonnie++, 2012] versão 2.3 para geração dos quatro (4) tipos de acesso concorrentes a disco: *Leitura-Aleatória*, *Escrita-Aleatória*, *Leitura-Sequencial* e *Escrita-Sequencial*. Para cada uma das cargas de trabalho, o *benchmark* bonnie ++ foi instruído para, repetidamente requisitar do sistema operacional operações 4 KB de leitura/escrita a partir de um arquivo armazenado no sistema de arquivos do disco virtual da MV. A cada requisição, o processo era mantido suspenso pelo sistema operacional até o recebimento de uma resposta à requisição. O tamanho do arquivo foi definido como sendo maior que a memória *cache* da MV.

Durante a realização dos experimentos que consideravam cargas de trabalho com acesso concorrente a disco, as sete (7) MVs (exceto a MV que executava o SGBDR) processavam simultaneamente e de forma idêntica uma instância do *benchmark* bonnie ++ devidamente adaptado conforme descrito.

5.4 Ambiente Experimental em Detalhes

A formalização apresentada na Tabela 5.3, busca enriquecer em detalhes a descrição do ambiente experimental e evitar equívocos ou imprecisão na apresentação dos resultados obtidos.

TABELA 5.3: FORMALIZAÇÃO DO AMBIENTE: NOTAÇÃO E DESCRIÇÃO

Notação	**Descrição**
R	Conjunto de recursos computacionais disponíveis no hospedeiro. $R=(p,\ a,\ d)$. p: valor numérico positivo medido em *Giga Hertz* (GHz) que define o total de processamento disponível. a: valor numérico positivo expresso em *Giga Bytes* (GB) que define o total de memória RAM disponível. d: valor numérico positivo expresso em Giga Bytes (GB) com taxa de transferência expressa em *Mega Bytes* por Segundo (MB/s) que define o total de espaço em disco disponível.
m	Número de recursos computacionais considerados. Nos experimentos, m=3.
$R=(r_{i1}\ \ldots\ r_{im})$	Recursos computacionais alocados de forma estática à máquina virtual, sendo $0 \leq\ r_{ij} \leq 1$.
MV_i	Representa a *i-ésima* máquina virtual criada e gerenciada pelo monitor de máquinas virtuais.
n	Número de máquinas virtuais disponíveis. Nos experimentos, n=8.
N	Conjunto de máquinas virtuais alocadas sobre um servidor físico, competindo por recursos. $N=(MV_1\ \ldots\ n)$.
z	Número de parâmetros de configuração do SGBDR a ser ajustado. Nos experimentos, z=3
δ_i	Domínio de valores conhecidos e aceitos para um parâmetro em específico do SGBDR.
D	Conjunto de parâmetros que serão configurados através do emprego de técnicas de *tuning*. D$=(d_1=\delta_1,\ \ldots\ ,\ d_z=\delta_z)$.
W	Carga de trabalho submetida ao SGBDR, composta por dezoito (18) consultas SQL.
O_i	Carga de trabalho de acesso a disco submetida à n-1 MVs, caracterizada pelas duas dimensões: *leitura/gravação* e *sequencial/aleatória*. O_i $(1 \leq i \leq n\text{-}1)$.
t_i	Tempo de execução da *i-ésima* consulta SQL representado em milissegundos *(ms)*
μ_i	Tempo médio de execução da *i-ésima* consulta SQL, representado em segundos (s). $\mu_i=(t_i1 + t_i2 + \ldots + t_i5)/5$.
μ_T	Tempo médio de execução de W representado em segundos (s). $\mu_T=((\mu_i + \mu_{i+1} + \ldots + \mu_{18}))/18$.

CAPÍTULO 6

DISCUSSÃO DOS RESULTADOS

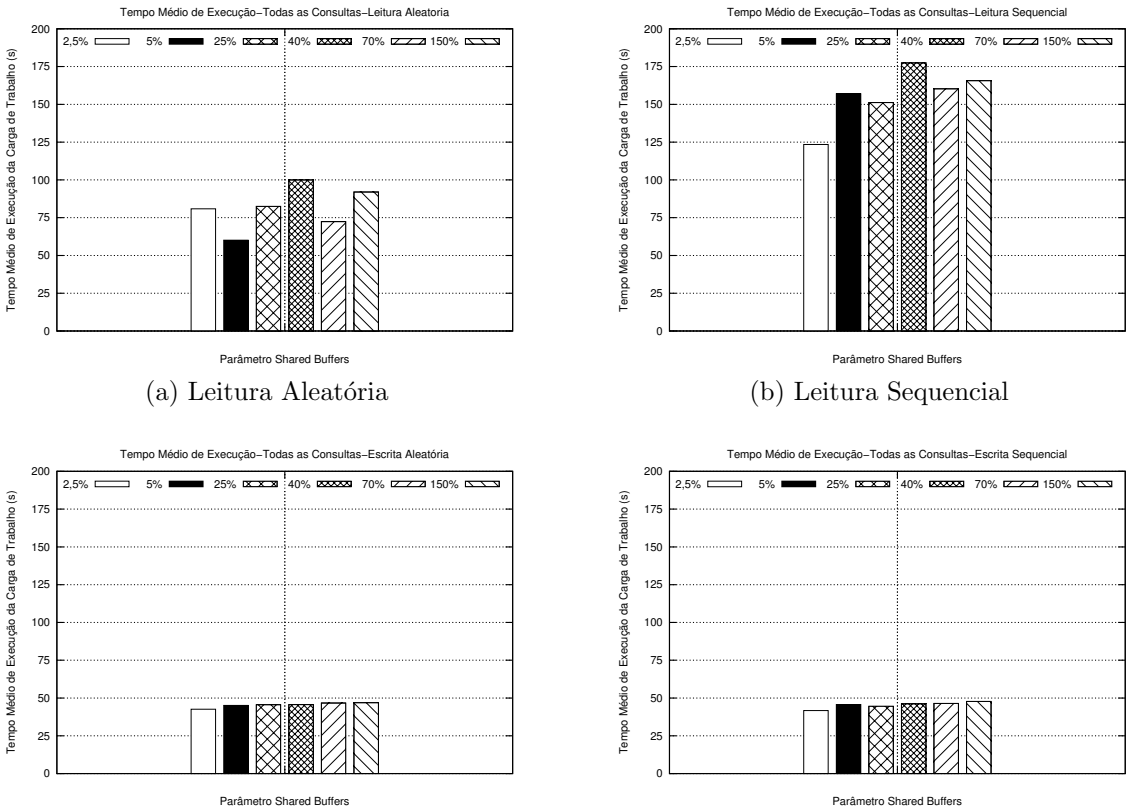
Este Capítulo apresenta e discute os resultados obtidos nos experimentos realizados. Por motivos de organização, o texto foi dividido em subseções de acordo com o número de parâmetros *(z)* de configuração do SGBDR analisado. Foi gerado um total de cento e noventa e cinco (195) gráficos nos experimentos realizados. Porém, devido à grande quantidade e espaço necessários tanto para suas representações quanto discussões, a apresentação destes torna-se totalmente inviável no contexto deste trabalho. Assim, a análise dará ênfase no tempo médio despendido para a execução completa da carga de trabalho SQL (μ_T). Portanto, de forma implícita, o tempo médio de execução de cada uma das dezoito (18) consultas SQL (μ_i) presentes em W tem influência no resultado apresentado [1]. Também iremos expor os dados obtidos pela execução de determinadas consultas em específico (μ_i), fomentando assim discussões e a extração de conclusões sobre seus resultados.

6.1 Parâmetro Shared_Buffers

Os resultados dos primeiros experimentos realizados levam em consideração o parâmetro de configuração *shared_buffers*. Ilustrados na Figura 6.1, representam em segundos, o tempo médio de execução da carga de trabalho SQL (μ_T). Cada um dos quatro (4) gráficos contempla a execução concorrente de um tipo de carga de trabalho de acesso a disco (O_i).

Considerando o acesso a disco categorizado como *Leitura-Aleatória*, que é ilustrado na Figura 6.1a, pode-se verificar que a regra de *tuning 40%* (regra-de-ouro) foi em média cerca de quarenta (40) segundos mais lenta comparada ao melhor desempenho, obtido

[1] Exceto para o parâmetro *work_mem*



(a) Leitura Aleatória

(b) Leitura Sequencial

(c) Escrita Aleatória

(d) Escrita Sequencial

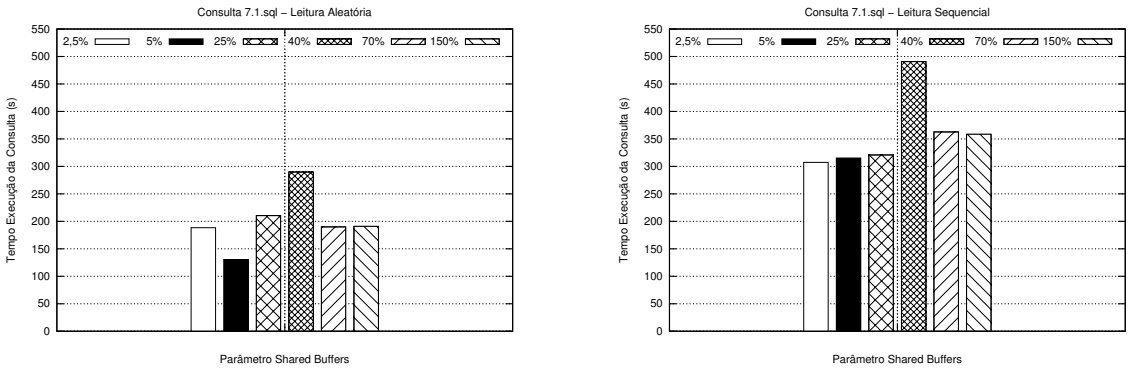
FIGURA 6.1: TEMPO MÉDIO DE EXECUÇÃO DA CARGA DE TRABALHO SQL CONSIDERANDO AS CARGAS DE ACESSO A DISCO E O *TUNING* NO PARÂMETRO *SHARED_BUFFERS*

pela regra *5%*. Outra regra de *tuning* que merece destaque é a *70%*, sendo em média cerca de doze (12) segundos mais lenta comparada a regra *5%*. As regras *2,5%* e *25%*, sendo esta última o valor padrão adotado pelo PostgreSQL, apresentaram tempos médios de execução muito próximos, com variação de menos de dois (2) segundos entre elas. Estas duas regras foram em média acima de vinte (20) segundos mais lentas comparadas a regra *5%*.

De forma semelhante, a regra *40%* não apresentou melhorias de desempenho para a carga de trabalho do tipo *Leitura-Sequencial* que é ilustrada na Figura 6.1b. Neste caso, tal regra levou cerca de cinquenta e cinco (55) segundos a mais para executar o mesmo conjunto de consultas SQL quando comparada a regra *2,5%* que obteve melhor desempenho. Esta última, foi em média cerca de vinte e oito (28) segundos mais rápida

para executar as consultas comparada à configuração padrão do PostgreSQL, representada pela regra *25%*.

Quanto a carga de trabalho de acesso a disco do tipo *Escrita-Sequencial* ilustrada na Figura 6.1d, o melhor resultado foi obtido pela regra *2,5%*, que executou as consultas SQL em média seis (6) segundos mais rápido comparado a regra *5%*, quatro (4) segundos comparada a regra *25%* (valor padrão do PostgreSQL) e sete (7) segundos comparada a regra *40%* caracterizada como regra-de-ouro. O acesso a disco categorizado como *Escrita-Aleatória* ilustrado na Figura 6.1c apresentou comportamento extremamente semelhante ao acesso concorrente do tipo *Escrita-Sequencial*. Em média, nenhum ganho de desempenho foi obtido aumentando consideravelmente os valores de configuração para o parâmetro *shared_buffers*, conforme mostra a regra *150%* em todos os gráficos da Figura 6.1. Enfim, alterações nas configurações padrão do parâmetro *shared_buffers* não são indicadas para acessos a disco do tipo *Escrita*, uma vez que apresentaram ganhos irrisórios, segundo as médias exibidas pelos experimentos através da Figura 6.1c e Figura 6.1d.



(a) Leitura Aleatória (b) Leitura Sequencial

FIGURA 6.2: TEMPO MÉDIO DE EXECUÇÃO DA CONSULTA *7.1.sql* CONSIDERANDO A CARGA DE ACESSO A DISCO E O *TUNING* NO PARÂMETRO *SHARED_BUFFERS*

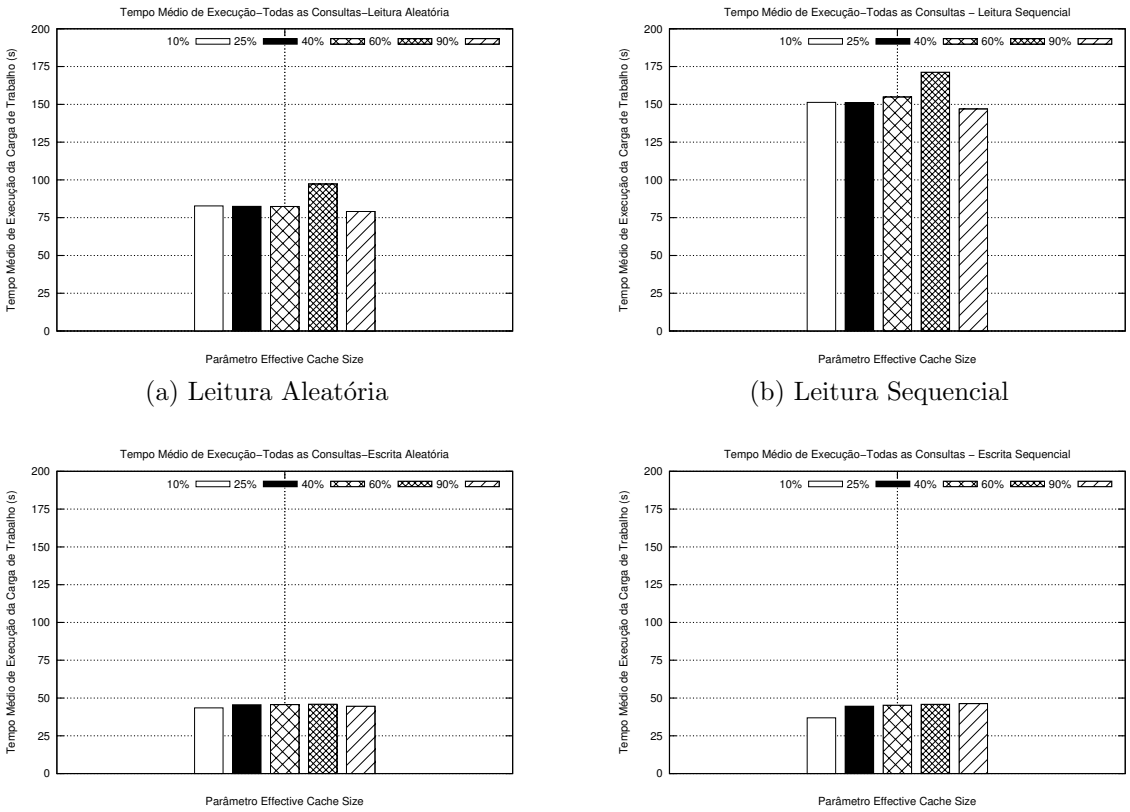
Ainda analisando o parâmetro *shared_buffers*, os experimentos demonstram como pode ser danoso ao desempenho do SGBDR seguir as regras-de-ouro, uma vez que suas definições não consideram a existência de cargas de trabalho de acesso a disco (O_i) concorrentes.

A Figura 6.2 demonstra esta situação, apresentando o tempo médio, em segundos, de execução da consulta (μ_i) *7.1.sql*, derivada do *benchmark* TPC-H, frente aos dois (2) tipos de acesso de *Leitura*. Como é possível observar na Figura 6.2b, a consulta executando sobre a regra *40%* demorou cerca de cento e oitenta e cinco (185) segundos a mais, para obter os resultados quando comparada a regra *2,5%*, que obteve o melhor desempenho considerando a carga de trabalho concorrente do tipo *Leitura-Sequencial*. De forma semelhante, ao analisar a carga do tipo *Leitura-Aleatória*, representada na Figura 6.2a, verifica-se que a regra *40%* retardou a exibição dos resultados em cerca de cento e sessenta (160) segundos quando comparada a regra *5%* com o melhor desempenho.

6.2 Parâmetro Effective_Cache_Size

Os gráficos que compõem a Figura 6.3 apresentam o tempo médio de execução, em segundos, da carga de trabalho composta pelas dezoito (18) consultas SQL *(μ_T)*, considerando o parâmetro *effective_cache_size*, sobre a influência dos quatro (4) tipos de cargas de trabalho de acesso a disco (O_i). Como observado, a regra *60%* (regra-de-ouro) não obteve bons resultados de desempenho, chegando aos piores, em operações de acesso a disco caracterizadas como *Leitura*. Isto é verificado principalmente no gráfico da Figura 6.3b, que ilustra o desempenho do SGBDR frente a carga de trabalho concorrente do tipo *Leitura-Sequencial*. Neste caso, a regra *60%* foi cerca de vinte e cinco (25) segundos mais lenta em relação a regra *90%* que obteve o melhor resultado de desempenho. Esse último resultado merece destaque, uma vez que não foi constatado similaridade em nenhuma média nos experimentos envolvendo o parâmetro *shared_buffers* como observado na Figura 6.1.

Não é possível verificar grandes variações de desempenho nos resultados apresentados pelas regras *10%* e *25%* considerando o acesso a disco do tipo *Leitura-Sequencial* conforme a Figura 6.3b. Quando observada a Figura 6.3a que apresenta o desempenho do SGBDR frente à carga de trabalho do tipo *Leitura-Aleatória* verifica-se grande semelhança de



(a) Leitura Aleatória

(b) Leitura Sequencial

(c) Escrita Aleatória

(d) Escrita Sequencial

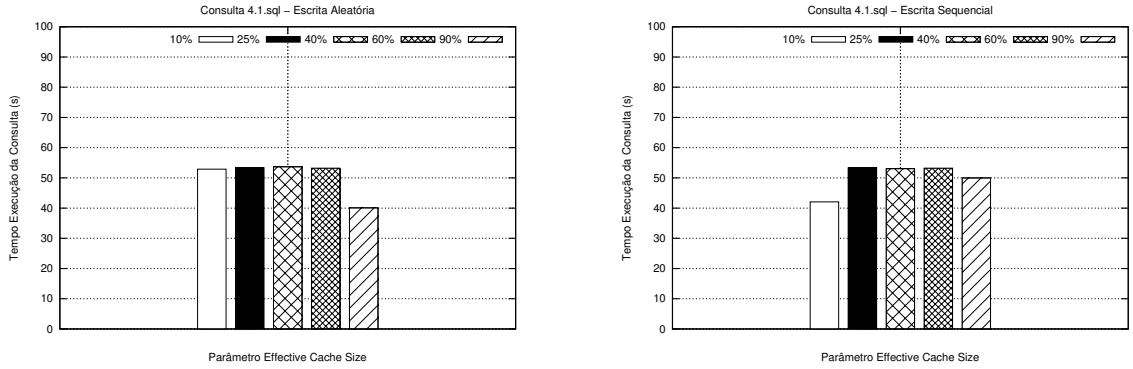
FIGURA 6.3: TEMPO MÉDIO DE EXECUÇÃO DA CARGA DE TRABALHO SQL CONSIDERANDO AS CARGAS DE ACESSO A DISCO E O *TUNING* NO PARÂMETRO *EFFECTIVE_CACHE_SIZE*

comportamento considerando o acesso concorrente a disco do tipo *Leitura-Sequencial*, logicamente que com menores tempos médios de execução do conjunto de consultas SQL. Neste caso a regra *90%* obteve desempenho ligeiramente melhor comparado às demais regras de *tuning* experimentadas. Sendo em média cerca de três (3) segundos mais rápida comparada a regra *40%* que obteve o segundo melhor resultado de desempenho.

Considerando-se o acesso concorrente a disco caracterizado como *Escrita* ilustrados nos gráficos das Figuras 6.3c e 6.3d, verifica-se que em ambos os casos o melhor desempenho foi obtido pelas regras *10%*. Principalmente para cargas de trabalho concorrentes do tipo *Escrita-Sequencial*, ilustrada na Figura 6.3d, que obteve médias de execução cerca de oito (8) segundos mais rápidas comparadas a regra *25%* que possui o valor padrão adotado pelo PostgreSQL. O SGBDR tornou-se menos eficiente utilizando-se as regras *60%* (regra-

de-ouro) e *90%* quando executado de forma concorrente com carga de trabalho do tipo *Escrita-Sequencial*.

De forma análoga, a regra *10%* sobre o parâmetro *effective_cache_size*, apresentou os melhores resultados de desempenho para operações de *Escrita-Aleatória* concorrentes a execução do SGBDR. Neste caso, a regra *10%* foi cerca de seis (6) segundos mais rápida comparada ao pior desempenho obtido pela regra *60%* (regra-de-ouro).



(a) Escrita Aleatória (b) Escrita Sequencial

FIGURA 6.4: TEMPO MÉDIO DE EXECUÇÃO DA CONSULTA *4.1.sql* CONSIDERANDO A CARGA DE ACESSO A DISCO E O *TUNING* NO PARÂMETRO *EFFECTIVE_CACHE_SIZE*

Finaliza-se a discussão a respeito do parâmetro *effective_cache_size* considerando os gráficos (a) e (b) na Figura 6.4. Estes demonstram o tempo médio, em segundos, de execução da consulta (μ_i) *4.1.sql*, frente aos acessos a disco concorrentes respectivamente do tipo *Escrita-Aleatória* e *Escrita-Sequencial*. Por meio dos resultados destes experimentos, constata-se a necessidade da verificação e estudo de novas regras de *tuning*, muitas vezes fora dos limites de valores indicados pelas regras-de-ouro e dos valores padrões adotados pelo SGBDR. Isto é comprovado analisando-se a regra *10%* ilustrada na Figura 6.4b, a qual foi cerca de oito (8) segundos mais veloz que a regra *90%* e doze (12) segundos mais rápida que a configuração *60%* caracterizada como regra-de-ouro, considerando a carga de trabalho concorrente de acesso a disco caracterizada como *Escrita-Sequencial*.

Outra constatação importante ao compararmos os gráficos da Figura 6.4 é de que o *tuning* deve ser realizado de acordo com a carga de trabalho de acesso a disco que

executa de forma concorrente ao SGBDR. Na Figura 6.4b que ilustra a carga de acesso a disco do tipo *Escrita-Sequencial*, o melhor desempenho em média para a consulta *4.1.sql* foi obtido pela regra *10%*, que se utiliza dos menores valores experimentados para o parâmetro *effective_cache_size*. Na Figura 6.4a que por sua vez ilustra a carga do tipo *Escrita-Aleatória*, de forma completamente oposta, o melhor desempenho foi obtido pela regra *90%*, ou seja, com o maior valor definido para o parâmetro *effective_cache_size*.

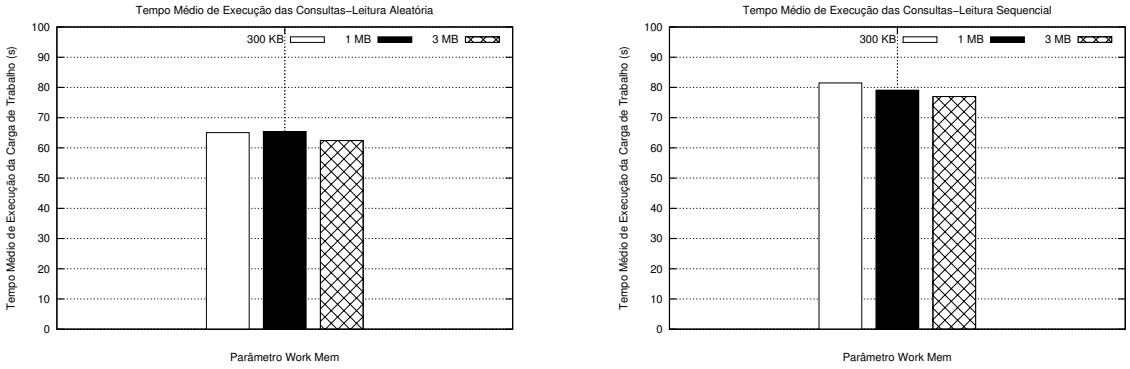
6.3 Parâmetro Work_Mem

Conforme já discutido, os valores utilizados para o parâmetro *work_mem*, limitam a quantidade de memória disponibilizada para operações de ordenação no SGBDR. Uma vez que as consultas que fazem uso de operações de ordenação representam apenas cerca de 30% do total de consultas da carga de trabalho SQL (W) utilizada, as mesmas não representam impacto significativo no tempo médio de sua execução (μ_T). Assim, exclusivamente para o parâmetro *work_mem*, efetuamos o cálculo do tempo médio, somente das consultas dessa porcentagem, o que é ilustrado na Figura 6.5. São representantes desse novo conjunto (30% do total) as consultas: *1.sql*, *2.1.sql*, *2.2.sql*, *4.2.sql*, *8.1.sql* e *16.2.sql*.

O que se verificou nestes experimentos é que o parâmetro *work_mem*, de maneira oposta aos parâmetros *effective_cache_size* e *shared_buffers*, possui baixa sensibilidade à aplicação de regras de *tuning* frente aos quatro (4) tipos de cargas de trabalho de acesso a disco. Esta afirmação é comprovada, verificando-se as escalas dos gráficos da Figura 6.5 que ao contrário das Figuras 6.3 e 6.1, tem seus valores fixados entre zero (0) e cem (100) segundos.

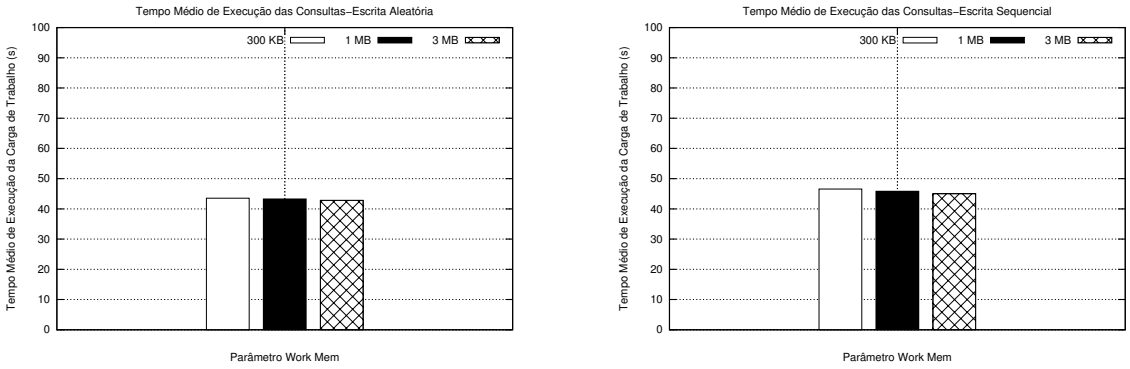
É possível observar uma pequena melhoria de desempenho para as cargas de trabalho concorrentes do tipo *Leitura*, quando aplicada a regra *3 MB* como mostra as Figuras 6.5a e 6.5b. Seus gráficos consideram respectivamente as cargas de acesso a disco caracterizadas como *Leitura Aleatória* e *Leitura Sequencial*. Nesta ordem, a regra *3 MB* foi três (3) segundos e dois (2) segundos mais rápida em relação a regra-de-ouro estipulada em *1 MB*.

Segundo os experimentos relacionados ao parâmetro *work_mem*, esforços para obtenção de boas definições de *tuning*, não se justificam devido à baixa variação na média dos tempos de execução da carga de trabalho SQL considerando os quatro (4) diferentes tipos de acessos concorrentes a disco. Tal constatação fica mais explícita nos gráficos das Figuras 6.5c e 6.5d, que consideram as cargas de acesso a disco do tipo *Escrita* onde as variações sequer são perceptíveis.



(a) Leitura Aleatória

(b) Leitura Sequencial



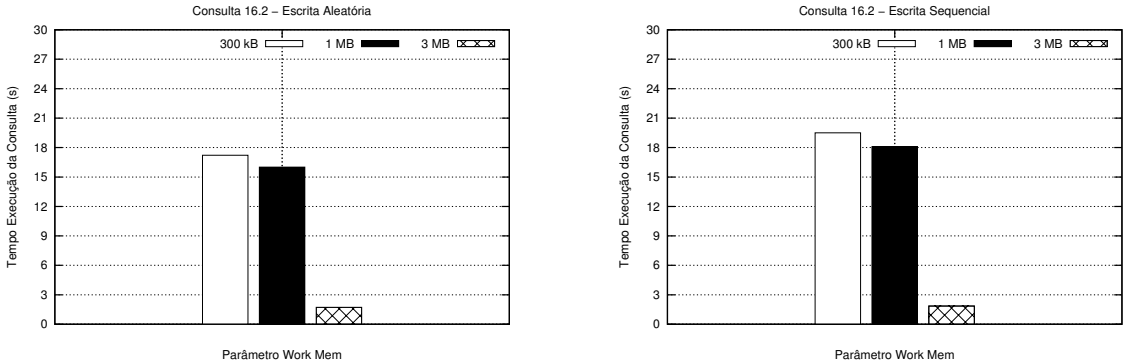
(c) Escrita Aleatória

(d) Escrita Sequencial

FIGURA 6.5: TEMPO MÉDIO DE EXECUÇÃO DA CARGA DE TRABALHO SQL CONSIDERANDO AS CARGAS DE ACESSO A DISCO E O *TUNING* NO PARÂMETRO *WORK_MEM*

Uma solução para obter melhorias de desempenho do SGBDR através do parâmetro *work_mem*, é a utilização de regras de *tuning* específicas para cada consulta, ou grupos de consultas, individualmente, considerando a carga de trabalho de acesso a disco que executa concorrentemente. Esta hipótese é comprovada pela Figura 6.6, que ilustra o tempo médio de execução da consulta (μ_i) *16.2.sql* que compõe a carga de trabalho SQL,

frente ao acesso à disco do tipo *Escrita*. Como é possível observar nas Figuras 6.6a e 6.6b, a configuração *1MB* não apresentou bons resultados de desempenho nos experimentos. Sendo quinze (15) e dezessete (17) segundos mais lenta na obtenção de resultados, em comparação ao melhor desempenho obtido pela regra *3MB*, respectivamente para cargas de trabalho de acesso a disco do tipo *Escrita Aleatória* e *Escrita Sequencial*.



(a) Escrita Aleatória (b) Escrita Sequencial

FIGURA 6.6: TEMPO MÉDIO DE EXECUÇÃO DA CONSULTA *16.2.sql* CONSIDERANDO A CARGA DE ACESSO A DISCO E O *TUNING* NO PARÂMETRO *WORK_MEM*

6.4 Discussão sobre os Melhores Resultados de Desempenho

Na tentativa de encontrar a configuração de *tuning* que apresente os melhores resultados de desempenho para cada uma das quatro (4) diferentes cargas de trabalho de acesso a disco, é realizada uma análise sobre o tempo médio de execução da carga de trabalho composta pelas dezoito (18) consultas SQL (μ_T) considerando os parâmetros *shared_buffers* e *effective_cache_size* que foram apresentados nas Figuras 6.1 e 6.3. Tal análise considerando o parâmetro *work_mem* não é possível ser efetuada, uma vez que a carga de trabalho submetida ao SGBDR em seus experimentos é distinta, pois conta apenas com 30% do total das dezoito (18) consultas SQL utilizadas.

Considerando a carga de trabalho de acesso a disco caracterizada como *Leitura-Aleatória*, Figura 6.1a e Figura 6.3a, constata-se que a regra de *tuning* mais eficiente

foi a *5%* sobre o parâmetro *shared_buffers* trazendo em média resultados em sessenta (60) segundos. A segunda regra mais bem posicionada foi a *70%* também sobre *shared_buffers* levando cerca de setenta e dois (72) segundos para exibir os mesmos resultados.

Quanto à carga de trabalho do tipo *Leitura-Sequencial*, exibida nas Figuras 6.1b e 6.3b, verifica-se que a regra *2,5%* sobre o parâmetro *shared_buffers* é a que obtém melhor desempenho. A mesma exibe a média de seus resultados em cento e vinte e três (123) segundos, vinte e três (23) segundos mais rapidamente em comparação a regra *90%* sobre o parâmetro *effective_cache_size* que obteve o segundo melhor resultado de tempo.

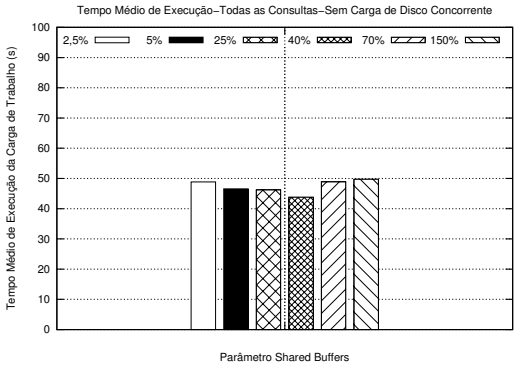
Em relação à carga de trabalho caracterizada como *Escrita-Aleatória*, apresentada nas Figuras 6.1c e 6.3c, a regra *10%* sobre o parâmetro *effective_cache_size* apresentou os melhores resultados de desempenho. Para a obtenção de resultados da carga de trabalho de consultas SQL foram necessários aproximadamente quarenta (40) segundos, sendo dois (2) segundos mais rápida em comparação ao segundo melhor resultado obtido pelas regras *2,5%* sobre os parâmetros *shared_buffers*.

A última análise se restringe à carga de trabalho do tipo *Escrita-Sequencial* que é ilustrada nas Figuras 6.1d e 6.3d. Neste caso, a regra *10%* sobre o parâmetro *effective_cache_size* obteve melhores resultados de desempenho. Um total de trinta e seis (36) segundos foi despendido para obter os resultados da carga de trabalho SQL. Esta regra de *tuning* foi cerca de cinco (5) segundos mais rápida comparada a regra *2,5%* sobre *shared_buffers*, a qual teve o segundo melhor resultado. Assim, verifica-se que valores menores inseridos no parâmetro *effective_cache_size* apresentam maior eficiência para cargas de trabalho do tipo *Escrita*.

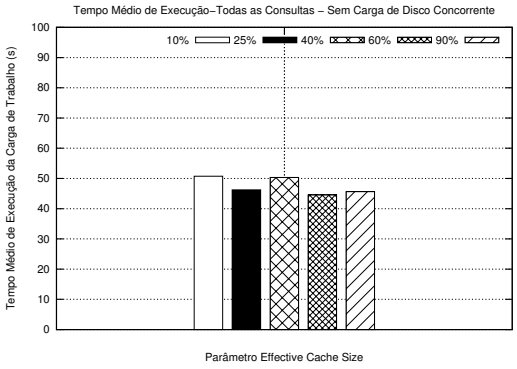
6.5 Resultados sem a Execução Concorrente de Cargas de Acesso a Disco

Conforme descrito, as regras-de-ouro, são recomendações para o *tuning* de SGBDRs comumente encontradas na literatura ou sugeridas por *experts*. Tais regras não levam em

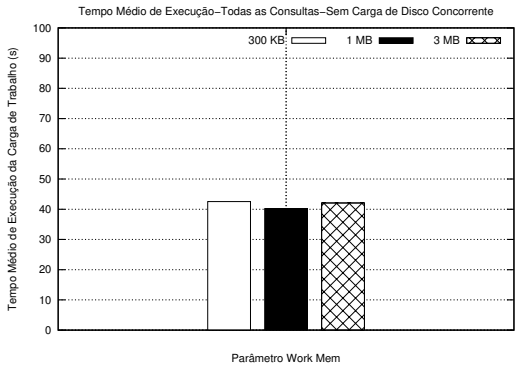
consideração a existência de cargas de trabalho oriundas de outras MVs que concorrem por recursos físicos junto ao SGBDR. Sendo assim, teoricamente elas são eficientes para ambientes dedicados. Buscando comprovar essa afirmação, os últimos experimentos a serem apresentados reproduzem tal situação, onde a MV que hospeda o SGBDR não sofre o acesso concorrente das cargas de trabalho de acesso a disco (O_i). Os dados destes experimentos que são ilustrados na Figura 6.7 seguem rigorosamente a mesma forma (μ_T) para obtenção utilizada nos demais experimentos que consideram as cargas concorrentes de acesso a disco.



(a) Parâmetro Shared Buffers



(b) Parâmetro Effective Cache Size



(c) Parâmetro Work Mem

FIGURA 6.7: TEMPO MÉDIO DE EXECUÇÃO DA CARGA DE TRABALHO SQL CONSIDERANDO *TUNING* NOS PARÂMETROS *SHARED_BUFFERS, EFFECTIVE_CACHE_SIZE e WORK_MEM*

Como é possível observar na Figura 6.7, as regras-de-ouro definidas como: (1) *40%* para *shared_buffers* (Figura 6.7a); (2) *60%* para *effective_cache_size* (Figura 6.7b) e (3) 1 MB para *wor_mem* (Figura 6.7c) foram, **em todos os casos**, mais eficientes em com-

paração as demais regras experimentadas. Estes resultados atestam suas validades quanto ao desempenho para ambientes dedicados. Quando comparados aos resultados das Figuras 6.1, 6.3 e 6.5, verifica-se a ineficiência destas, uma vez que, **em nenhum dos experimentos** que consideram cargas de trabalho concorrentes de acesso a disco, tais regras foram capazes de obter os melhores resultados de desempenho.

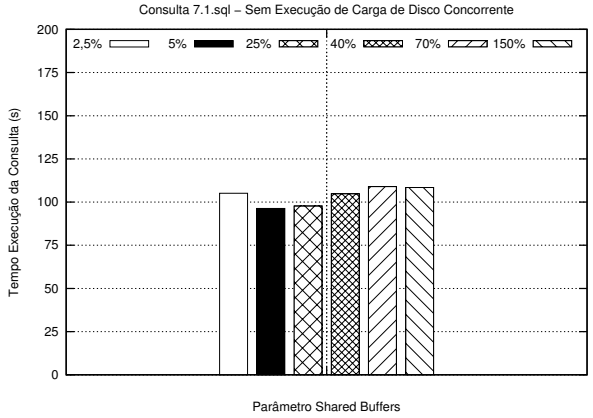


FIGURA 6.8: TEMPO MÉDIO DE EXECUÇÃO DA CONSULTA *7.1.sql* CONSIDERANDO *TUNING* NO PARÂMETRO *SHARED_BUFFERS*

Outro resultado que merece destaque é apresentado na Figura 6.8. Nela temos os dados sobre o tempo médio de execução da consulta (μ_i) *7.1.sql* da carga de trabalho SQL. É considerado o *tuning* sobre o parâmetro *shared_buffers* e a ausência de cargas de trabalho de acesso a disco, executantes de forma concorrente ao SGBDR. Constata-se que nenhuma regra de *tuning* utilizada, mais especificamente a regra *40%* (regra-de-ouro), sofreram grandes variações em suas médias de tempo de execução. Fato este não observado nas Figuras 6.2a e 6.2b quando o SGBDR estava executando concorrentemente frente às cargas de acesso a disco do tipo *Leitura* com várias configurações sobre o parâmetro *shared_buffers*. Nos experimentos relacionados ao tipo de acesso a disco caracterizado como *Leitura Sequencial*, variações de cento e oitenta e cinco (185) segundos ocorreram comparando a regra-de-ouro (regra *40%*) com o melhor resultado obtido pela regra *2,5%*. Isto demonstra que a atividade de *tuning* é de grande importância para o bom desempenho de SGBDR. Mais ainda, tal atividade é extremamente complexa, requer profundos conhecimentos e seus efeitos, benéficos ou não, são potencializados quando se trata de

ambientes computacionais virtualizados.

6.6 Novas Regras-de-Ouro para o Tuning de SGBDR em Ambientes Virtualizados

A partir dos resultados dos experimentos, demonstrados nas figuras das subseções 6.1 e 6.2, é possível fazer o apontamento de novas regras-de-ouro aplicadas ao *tuning* de SGBDRs quando inseridos em ambientes virtualizados. Essas regras são categorizadas de acordo com o tipo das cargas de trabalho de acesso a disco que concorrem por recursos junto ao SGBDR.

Para a carga de trabalho de acesso a disco do tipo *Leitura-Aleatória* a nova regra de *tuning* indicada pelo experimento é a aplicação de valores que representem cerca de *5%* da memória RAM disponível na MV para o parâmetro *shared_buffers*. Quanto ao parâmetro *effective_cache_size* nossos experimentos demontram a regra que se utiliza de 90% da memória RAM ser mais eficiênte.

Quanto à carga de trabalho caracterizada como *Leitura-Sequencial* recomenda-se a utilização de valores que correspondam a cerca de *2,5%* do total da memória RAM disponível na MV para o parâmetro *shared_buffers*. Bons resultados são obtidos com o parâmetro *effective_cache_size*, disponibilizando-lhe valores que representem cerca de *90%* do total de RAM disponível na MV.

Uma regra de *tuning* que traz benefícios de desempenho ao SGBDR, considerando a carga de trabalho do tipo *Escrita-Aleatória* é atribuir valores que representem cerca de *10%* da RAM disponível na MV ao parâmetro *effective_cache_size*. Outra regra que alcançou melhorias de desempenho é a atribuição de valores que equivalem a aproximadamente *2,5%* do total da RAM da MV ao parâmetro *shared_buffers*.

Finalmente, atribuir valores que correspondam a cerca de *10%* do total de RAM da MV ao parâmetro *effective_cache_size* e *2,5%* do total da RAM da MV ao parâmetro *shared_buffers*, traz benefícios ao desempenho do SGBDR que concorre em acesso a disco

com uma carga de trabalho caracterizada como *Escrita-Sequencial*. Sendo assim, para operações de *Escrita*, tanto *Sequencial* quanto *Aleatória* recomenda-se o uso das mesmas novas regras-de-ouro para o *tuning* do SGBDR.

É importante destacar que a utilização em conjunto das novas regras-de-ouro mencionadas, pode não trazer benefícios ao desempenho do SGBDR. Um exemplo é o emprego simultâneo da regra *5%* sobre o parâmetro *shared_buffers* e *25%* sobre o parâmetro *effective_cache_size* quando ocorrer a execução de cargas de trabalho do tipo *Leitura-Aleatória* concorrentes ao SGBDR. Atestar que tais configurações sejam eficientes trazendo melhorias ao desempenho do SGBDR exige grande quantidade de testes e análises.

Salienta-se que fatores como alterações na quantidade de MVs que executam sobre o hospedeiro, assim como mudanças nas configurações de hardware, tanto do sistema hospedeiro quanto das MVs podem resultar na definição de novas regras-de-ouro para o *tuning* do SGBDR em questão.

As novas regras-de-ouro apontadas podem servir de base para o desenvolvimento de uma ferramenta que permita automatizar o processo de *tuning* de SGBDRs em ambientes virtualizados. Esta ferramenta deverá efetuar análises identificando o tipo de carga de acesso a disco que concorrem junto ao SGBDR, e a partir disso realizar a configuração da aplicação de banco de dados *on-the-fly*, utilizando as regras-de-ouro ora apontadas ou outras resultantes de demais pesquisas. Esta solução pode trazer grandes benefícios de desempenho ao SGBDR.

CAPÍTULO 7

CONCLUSÃO E TRABALHOS FUTUROS

A tecnologia de virtualização apresenta inúmeras vantagens que tem estimulado sua adoção, como a racionalização de recursos e a flexibilidade administrativa dos serviços de TI. Contudo, quando inserimos SGBDRs em ambientes virtualizados é importante observar que somente o particionamento de recursos não é o suficiente para garantir a eficiência de seu desempenho. Esta tese comprovou esta afirmação, através de uma avaliação experimental que simula diferentes cargas de trabalho de acesso a disco concorrentes à execução do SGBDR. Verificou-se que o emprego das regras-de-ouro, geralmente aplicadas a sistemas que não competem por recursos físicos, através de técnicas de *tuning*, não resultam na maioria dos casos, em benefícios ao desempenho do SGBD que executa sobre o modelo *shared-hardware* de virtualização.

Os resultados também demonstraram que qualquer esforço, aplicando regras de *tuning* em um determinado parâmetro de configuração do SGBDR pode ser em vão. Isto se deve à característica da carga de trabalho de acesso a disco que compete por este recurso. Como exemplo, pode-se citar as cargas do tipo *Escrita*, onde nos experimentos foram alcançados pequenos ganhos de desempenho na execução do conjunto de consultas SQL. Tal fato se deve em grande parte à intensa exigência feita às unidades físicas de armazenamento, sendo consideradas o "gargalo" no desempenho do SGBDR. Esforços no emprego e testes de regras de *tuning* sobre parâmetros que não apresentam sensibilidade às consultas SQL submetidas também podem tornar-se frustrantes. Foi o que os experimentos demonstraram considerando o *parâmetro work_mem*. Independente das configurações adotadas pelas regras de *tuning* a ele submetidas, não ocorreram variações significativas em seus resultados considerando a existência dos quatro (4) tipos de cargas de trabalho de acesso a disco.

O processo de *tuning* de um SGBDR é extremamente complexo, devido à grande variedade de parâmetros a serem estudados e considerados para a sua devida configuração. São exigidos do profissional de banco de dados profundos conhecimentos a respeito da plataforma operacional utilizada, dos recursos computacionais disponíveis, do esquema do banco de dados e da carga de trabalho a ele submetida. Quando inserimos o SGBDR em um ambiente virtualizado, tal complexidade é potencializada acrescentando mais uma variável de incerteza: o tipo de carga de trabalho de acesso a disco que executa concorrente ao SGBD.

O resultado exibido pela consulta *7.1.sql* frente aos acesso concorrentes do tipo *Leitura-Sequencial* é um exemplo muito interessante. Ao aplicarmos uma regra de *tuning* na tentativa de obter melhorias no desempenho do SGBDR, fez com que o resultado demorasse cerca de três (3) minutos a mais para ser exibido em comparação a outra regra, que sequer possui recomendação de seu uso na literatura. Esses fatos comprovam que há existência de regras de *tuning* especificas, que trarão melhores resultados de desempenho para SGBDRs alocados em ambientes elásticos. Regras estas, muitas vezes bastante distintas das recomendadas para sistemas dedicados, que se caracterizam como novas regras-de-ouro para o *tuning* de SGBDRs inseridos em ambientes virtualizados.

Os SGBDRs geralmente atendem às requisições de diversos tipos de sistemas legados, que certamente ainda estarão em operação por longo período de tempo nas organizações. A inserção desses sistemas de banco de dados no ambiente de computação em nuvem, mais especificamente através do modelo *shared-hardware* é uma necessidade de grande relevância como forma de redução de custos operacionais. Assim, é evidente a exigência de futuras pesquisas relacionadas à otimização de seu desempenho. As seguintes contribuições são proporcionadas por este trabalho:

- Desenvolvimento de um método para análise da eficiência das regras-de-ouro aplicadas a SGBDRs em ambientes virtualizados. Para isso fez-se necessário:
  - Caracterizar os principais tipos de acesso às unidades de disco;

  - Adaptar a ferramenta de *benchmark* que analisa o desempenho e o comportamento de discos rígidos e sistemas de arquivos;
  - Adaptar o conjunto de consultas SQL derivadas do *benchmark* TPC-H para compor a carga de trabalho submetida ao banco de dados;
  - Delimitar o conjunto de parâmetros de configuração do SGBDR que apresente maior influência dado o esquema e a carga de trabalho de consultas SQL;
  - Capturar e analisar o tempo médio de execução de cada consulta individualmente, além do tempo médio total para execução da carga de trabalho composta pelas consultas SQL.

- Comprovação da inadequação das tradicionais regras-de-ouro para o *tuning* de SGBDRs inseridos em ambientes virtualizados;

- Apontamento de novas regras-de-ouro para o *tuning* de SGBDRs inseridos em ambientes virtualizados, de acordo com as cargas de trabalho de acesso a disco que executam concorrentemente, considerando também as configurações de hardware e software disponíveis.

7.1 Trabalhos Futuros

Os trabalhos futuros são classificados em três grupos de atuação considerando a inserção de SGBDRs em ambientes virtualizados do tipo *shared-hardware*: (1) realização de experimentos sobre outros parâmetros de configuração, (2) automatização do processo de *tuning* e (3) adaptação do SGBDR para torná-lo apto a operar de forma eficiente em ambientes de computação em nuvem.

Os experimentos desenvolvidos nesta tese ficaram restritos a três parâmetros de configuração que foram analisados tomando como base o *benchmark* TPC-H que simula um ambiente OLAP. Um SGBDR típico apresenta diversos parâmetros que podem ser examinados sendo possível sobre estes, experimentar diversas regras de *tuning*. Assim, analisar

valores para os parâmetros de configuração do SGBDR considerando cargas de trabalho transacionais do tipo OLTP, resultarão em novas regras de *tuning* aplicadas a outros parâmetros de configuração, trazendo benefícios ao desempenho de SGBDR em ambientes do tipo *shared-hardware*. Ainda neste sentido, uma análise criteriosa pode ser realizada sobre os parâmetros caracterizados como *descriptive parameters*, experimentando regras de *tuning* que possam trazer benefícios para a escolha de planos de execução de consultas de menor custo.

Outro trabalho a ser realizado é a concepção de uma ferramenta que realize tanto a análise dos tipos de acessos a disco concorrentes, quanto das consultas SQL em execução. Com base nestas informações e juntamente com a disponibilidade de recursos físicos, proponha a aplicação de regras de *tuning* ao SGBDR (*on-the-fly*). Esta ferramenta seria um complemento importante ao trabalho de Soror *et al.* [Soror et al., 2008], de forma que, além do gerenciamento de recursos físicos entre as máquinas virtuais ser realizado de forma contínua, o SGBDR estará apto a responder de forma mais precisa às requisições impostas, frente às constantes variações de recursos e cargas de trabalho comuns em ambientes de computação em nuvem.

Conforme já discutimos, outra maneira de otimizar o desempenho de SGBDRs é realizando intervenções nas etapas que compreendem o processamento de consultas. Um trabalho de grande relevância é o de adaptar o SGBDR de modo que seus mecanismos de auto-configuração sejam capazes de reconhecer o dinamismo do provisionamento de recursos gerado pelo ambiente de computação em nuvem. Para isso, uma nova arquitetura de custos deve ser concebida definindo as informações trocadas entre o MMVs e o SGBDR. Dessa forma, o modelo de custos terá informações atualizadas sobre a disponibilidade de recursos acarretando em maior precisão de suas estimativas, gerando planos de execução de consultas mais eficientes que resultam em melhor desempenho para o banco de dados.

APÊNDICE A

CONSULTAS SQL ADAPTADAS DO BENCHMARK TPC-H

```
1  ------------------ Consulta: 1.sql ------------------

   select
           l_returnflag,
           l_linestatus,
           sum(l_quantity) as sum_qty,
           sum(l_extendedprice) as sum_base_price,
           sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
           sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
           avg(l_quantity) as avg_qty,
           avg(l_extendedprice) as avg_price,
           avg(l_discount) as avg_disc,
           count(*) as count_order
   from
           lineitem
   where
           l_shipdate <= date'1998-12-01' - interval '93 days'
   group by
           l_returnflag,
           l_linestatus
   order by
           l_returnflag,
           l_linestatus;

   ------------------ Consulta: 2.1.sql ------------------

   select
           s_acctbal,
           s_name,
           s_address,
           s_phone,
           s_comment
   from
           supplier
   order by
           s_acctbal desc,
           s_name
   LIMIT 100;

   ------------------ Consulta: 2.2.sql ------------------

   select
           n_name
   from
           nation

   order by
           n_name

   LIMIT 100;

   ------------------ Consulta: 2.3.sql ------------------

   select
           p_partkey,
           p_mfgr
   from
           part
```

```
where
        p_size = 34
        and p_type like '%STEEL'
LIMIT 100;

------------------ Consulta: 4.1.sql ------------------

select  *

from lineitem

where
        l_commitdate < l_receiptdate;

------------------ Consulta: 4.2.sql ------------------

select
        o_orderpriority,
        count(*) as order_count
from
        orders
where
        o_orderdate >= date '1994-11-01'
        and o_orderdate < date '1994-11-01' + interval '3 month'
group by
        o_orderpriority
order by
        o_orderpriority;

------------------ Consulta: 5.1.sql ------------------

select n_name
from
        nation

group by
        n_name;

------------------ Consulta: 6.sql ------------------

select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= date '1993-01-01'
        and l_shipdate < date '1993-01-01' + interval '1 year'
        and l_discount between 0.07 - 0.01 and 0.07 + 0.01
        and l_quantity < 24;

------------------ Consulta: 7.1.sql ------------------

select
                        extract(year from l_shipdate) as l_year,
                        l_extendedprice * (1 - l_discount) as volume
                from
                        lineitem
                where
                        l_shipdate between date '1995-01-01' and date '1996-12-31';

------------------ Consulta: 8.1.sql ------------------

select
        extract(year from o_orderdate) as o_year
from
        orders
where
        o_orderdate between date '1995-01-01' and date '1996-12-31'
group by
        o_year
```

```
order by
        o_year;

------------------ Consulta: 13.1.sql ------------------

select
        count(o_orderkey)
from
        orders
where
        o_comment not like '%pending%deposits%';


------------------ Consulta: 15.1.sql ------------------

select
      l_suppkey,
      sum(l_extendedprice * (1 - l_discount))
from
      lineitem
where
      l_shipdate >= '1997-10-01'
      and l_shipdate < date'1997-10-01' + interval '90 days'
group by
      l_suppkey;

------------------ Consulta: 16.1.sql ------------------

select
        s_suppkey
from
        supplier
where
        s_comment like '%Customer%Complaints%';

------------------ Consulta: 16.2.sql ------------------

select
        p_brand,
        p_type,
        p_size
from
        part
where
        p_brand <> 'Brand#25'
        and p_type not like 'SMALL POLISHED%'
        and p_size in (34, 16, 19, 38, 20, 45, 8, 41)
group by
        p_brand,
        p_type,
        p_size
order by
        p_brand,
        p_type,
        p_size;

------------------ Consulta: 17.1.sql ------------------

select
        0.2 * avg(l_quantity)
from
        lineitem;

------------------ Consulta: 18.1.sql ------------------

select
        l_orderkey
from
        lineitem
group by
```

```
        l_orderkey
 having
        sum(l_quantity) > 313;

------------------ Consulta: 19.1.sql ------------------

select
        sum(l_extendedprice* (1 - l_discount)) as revenue
from
        lineitem
where
        l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
        and
        (
         (
                l_quantity >= 2 and l_quantity <= 2+10
         )
         or
         (
                l_quantity >= 18 and l_quantity <= 18+10
          )
          or
          (
                l_quantity >= 28 and l_quantity <= 28+10
          )
        );

------------------ Consulta: 20.1.sql ------------------

select
        0.5 * sum(l_quantity)
from
        lineitem
where
        l_shipdate >= '1994-01-01'
        and l_shipdate < date '1994-01-01' + interval '1 year';
```

APÊNDICE B

TRABALHOS DESENVOLVIDOS SOBRE OTIMIZAÇÃO DE DESEMPENHO DE SGBDR

A lista a seguir apresenta os trabalhos desenvolvidos sobre a otimização de desempenho de SGBDRs:

1. Tarcizio Alexandre Bini, Adriano Lange, Marcos Sfair Sunye, e Fabiano Silva. Stableness in large join query optimization. ISCIS, páginas 639-644, 2009.

2. Tarcizio Alexandre Bini, Adriano Lange, Marcos Sfair Sunye, Fabiano Silva, e Eduardo Cunha de Almeida. Non-exhaustive Join Ordering Search Algorithms for LJQO. ICEIS, páginas 151-156, 2011.

3. Simone Dominico, Tarcizio Alexandre Bini. Tuning: Um Estudo sobre a Otimização de Desempenho de SGBDR sob Cargas de Trabalho Transacionais e de Suporte a Decisão. I Encontro de Computação e Matemática Aplicada. Universidade Tecnológica Federal do Paraná, 2013.

4. Tarcizio Alexandre Bini, Adriano Lange, Marcos Sfair Sunye. Cloud Computing: An Evaluation of Rules of Thumb for Tuning RDBMs. ICEIS, 2014.

REFERÊNCIAS

[Azu, 2012] (2012). Windows Azure: Cloud Computing — Cloud Services — Cloud Application Development. Disponível em: `http://www.windowsazure.com/en-us/`, acessado em 19/09/2012.

[ama, 2013] (2013). Amazon elastic compute cloud (amazon ec2). Disponível em: `http://www.aws.amazon.com/ec2/`, acessado em 10/11/2013.

[Agrawal et al., 2012] Agrawal, D., Das, S., and El Abbadi, A. (2012). *Data Management in the Cloud: Challenges and Opportunities*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[Alashqur et al., 1989] Alashqur, A. M., Su, S. Y. W., and Lam, H. (1989). Oql: a query language for manipulating object-oriented databases. In *Proceedings of the 15th international conference on Very large data bases*, VLDB '89, pages 433–442, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Amazon Web Services, 2013] Amazon Web Services (2013). Amazon web services (aws). Disponível em: `http://aws.amazon.com/`, acessado em 10/11/2013.

[Astrahan and Chamberlin, 1975] Astrahan, M. M. and Chamberlin, D. D. (1975). Implementation of a structured english query language. *Commun. ACM*, 18:580–588.

[Aulbach et al., 2008] Aulbach, S., Grust, T., Jacobs, D., Kemper, A., and Rittinger, J. (2008). Multi-tenant databases for software as a service: schema-mapping techniques. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1195–1206, New York, NY, USA. ACM.

[Babcock and Chaudhuri, 2005] Babcock, B. and Chaudhuri, S. (2005). Towards a robust query optimizer: a principled and practical approach. In *Proceedings of the 2005 ACM*

SIGMOD international conference on Management of data, SIGMOD '05, pages 119–130, New York, NY, USA. ACM.

[Baker et al., 2011] Baker, J., Bond, C., Corbett, J. C., Furman, J., Khorlin, A., Larson, J., Leon, J.-M., Li, Y., Lloyd, A., and Yushprakh, V. (2011). Megastore: Providing scalable, highly available storage for interactive services. In *Proceedings of the Conference on Innovative Data system Research (CIDR)*, pages 223–234.

[Bennett, 1995] Bennett, K. (1995). Legacy systems: Coping with success. *IEEE Softw.*, 12(1):19–23.

[Bini et al., 2009] Bini, T. A., Lange, A., Sunyé, M. S., and Silva, F. (2009). Stableness in large join query optimization. In *ISCIS*, pages 639–644.

[Bini et al., 2011] Bini, T. A., Lange, A., Sunyé, M. S., Silva, F., and Almeida, E. C. d. (2011). Non-exhaustive Join Ordering Search Algorithms for LJQO. In *ICEIS (1)*, pages 151–156.

[Blue Cloud, 2013] Blue Cloud (2013). Ibm introduces ready-to-use cloud computing. Disponível em: `http://www-03.ibm.com/press/us/en/pressrelease/22613.wss`, acessado em 12/10/2013.

[Bonnie++, 2012] Bonnie++ (2012). Bonnie++ benchmark. Disponível em: `http://www.coker.com.au/bonnie++/`, acessado em 06/06/2012.

[Brodie and Stonebraker, 1995] Brodie, M. L. and Stonebraker, M. (1995). *Migrating Legacy Systems: Gateways, Interfaces, and the Incremental Approach*. Morgan Kaufmann.

[Bruno, 2003] Bruno, N. (2003). *Statistics on query expressions in relational database management systems*. PhD thesis, Columbia University, New York, NY, USA. AAI3088302.

[Buyya et al., 2009] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616.

[Carissimi, 2008] Carissimi, A. (2008). *Mini-cursos do SBRC 2008*, chapter Virtualização: da Teoria a Soluções, pages 173 – 207. Sociedade Brasileira de Computação.

[Cattell, 2011] Cattell, R. (2011). Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27.

[Chaudhuri, 1998] Chaudhuri, S. (1998). An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pages 34–43, New York, NY, USA. ACM.

[Chieu et al., 2009] Chieu, T. C., Mohindra, A., Karve, A. A., and Segal, A. (2009). Dynamic scaling of web applications in a virtualized cloud computing environment. In *Proceedings of the 2009 IEEE International Conference on e-Business Engineering*, ICEBE '09, pages 281–286, Washington, DC, USA. IEEE Computer Society.

[Clustrix, 2012] Clustrix (2012). Clustrix: Speed. Scale. Simplicity. Disponível em: `http://www.clustrix.com/`, acessado em 19/01/2012.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387.

[Codd, 1972] Codd, E. F. (1972). Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California.*

[CRM Salesforce, 2013] CRM Salesforce (2013). Crm salesforce e a computação nas nuvens para expandir seus negócios. Disponível em: `http://www.salesforce.com/`, acessado em 12/11/2013.

[Curino et al., 2011] Curino, C., Jones, E., Popa, R. A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., and Zeldovich, N. (2011). Relational cloud: a database service for the cloud. In *CIDR*, pages 235–240.

[Debnath et al., 2008a] Debnath, B. K., Lilja, D. J., and Mokbel, M. F. (2008a). Exploiting the impact of database system configuration parameters: A design of experiments approach. volume 31, pages 3–10.

[Debnath et al., 2008b] Debnath, B. K., Lilja, D. J., and Mokbel, M. F. (2008b). Sard: A statistical approach for ranking database tuning parameters. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering Workshop*, ICDEW '08, pages 11–18, Washington, DC, USA. IEEE Computer Society.

[Delimitrou et al., 2012] Delimitrou, C., Sankar, S., Khessib, B., Vaid, K., and Kozyrakis, C. (2012). Time and cost-efficient modeling and generation of large-scale tpcc/tpce/tpch workloads. In *Proceedings of the Third TPC Technology conference on Topics in Performance Evaluation, Measurement and Characterization*, TPCTC'11, pages 146–162, Berlin, Heidelberg. Springer-Verlag.

[Dias et al., 2005] Dias, K., Ramacher, M., Shaft, U., Venkataramani, V., and Wood, G. (2005). Automatic performance diagnosis and tuning in oracle. In *CIDR*, pages 84–94.

[Duan et al., 2009] Duan, S., Thummala, V., and Babu, S. (2009). Tuning database configuration parameters with ituned. *Proc. VLDB Endow.*, 2(1):1246–1257.

[Elmore et al., 2011] Elmore, A., Das, S., Agrawal, D., and Abbadi, A. E. (2011). Towards an elastic and autonomic multitenant database. In *NetDB 2011 - 6th International Workshop on Networking Meets Databases Co-located with SIGMOD 2011.*

[Floratou et al., 2012] Floratou, A., Teletia, N., DeWitt, D. J., Patel, J. M., and Zhang, D. (2012). Can the elephants handle the nosql onslaught? *Proc. VLDB Endow.*, 5(12):1712–1723.

[Garcia-Molina et al., 2008] Garcia-Molina, H., Ullman, J. D., and Widom, J. (2008). *Database Systems: The Complete Book*, chapter The Query Compiler, pages 759–841. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition.

[Google App Engine, 2013] Google App Engine (2013). Google app engine - google developers. Disponível em: `https://developers.google.com/appengine/`, acessado em 12/10/2013.

[Gulati et al., 2010] Gulati, A., Kumar, C., and Ahmad, I. (2010). Modeling workloads and devices for IO load balancing in virtualized environments. *SIGMETRICS Perform. Eval. Rev.*, 37(3):61–66.

[Gupta et al., 2006] Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A. (2006). Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 342–362, New York, NY, USA. Springer-Verlag New York, Inc.

[Harizopoulos et al., 2008] Harizopoulos, S., Abadi, D. J., Madden, S., and Stonebraker, M. (2008). Oltp through the looking glass, and what we found there. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 981–992, New York, NY, USA. ACM.

[Heroku, 2013] Heroku (2013). Heroku dev center. Disponível em: `https://devcenter.heroku.com/`, acessado em 12/10/2013.

[Hsu et al., 2001] Hsu, W. W., Smith, A. J., and Young, H. C. (2001). I/o reference behavior of production database workloads and the tpc benchmarks an analysis at the logical level. *ACM Trans. Database Syst.*, 26(1):96–143.

[Hui et al., 2009] Hui, M., Jiang, D., Li, G., and Zhou, Y. (2009). Supporting database applications as a service. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 832–843, Washington, DC, USA. IEEE Computer Society.

[IBM z/VM, 2013] IBM z/VM (2013). IBM: z/VM Operating System. Disponível em: `http://www.vm.ibm.com/`, acessado em 15/11/2013.

[Ioannidis, 1996] Ioannidis, Y. E. (1996). Query optimization. *ACM Comput. Surv.*, 28:121–123.

[Jacobs et al., 2007] Jacobs, D., Aulbach, S., and München, T. U. (2007). Ruminations on multi-tenant databases. In *BTW Proceedings, volume 103 of LNI*, pages 514–521. GI.

[Kvm, 2013] Kvm (2013). Kernel Based Virtual Machine. Disponível em: `http://www.linux-kvm.org/`, acessado em 15/11/2013.

[Lange, 2010] Lange, A. (2010). Uma avaliação de algoritmos não exaustivos para a otimização de junções. Dissertação de mestrado, Departamento de Informática, UFPR.

[Lazarov, 2007] Lazarov, V. (2007). *Comparison of Different Implementations of Multi-Tenant Databases*. PhD thesis, Technische Universit at Munchen.

[Maziero, 2013] Maziero, C. A. (2013). *Sistemas Operacionais: Conceitos e Mecanismos*, chapter Virtualização de Sistemas, pages 295 –331. DAINF - UTFPR.

[Mc Evoy et al., 2011] Mc Evoy, G. V., Schulze, B., and Garcia, E. L. M. (2011). Performance and deployment evaluation of a parallel application on a private cloud. *Concurr. Comput. Pract. Exper.*, 23(17):2048–2062.

[MySql, 2012] MySql (2012). MySQL : The world's most popular open source database. Disponível em: `http://www.mysql.com`, acessado em 18/01/2012.

[MySQL Cluster, 2012] MySQL Cluster (2012). MySQL : MySQL Cluster CGE. Disponível em: `http://www.mysql.com/products/cluster/`, acessado em 18/01/2012.

[Narayanan et al., 2005] Narayanan, D., Thereska, E., and Ailamaki, A. (2005). Continuous resource monitoring for self-predicting dbms. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '05, pages 239–248, Washington, DC, USA. IEEE Computer Society.

[Nobile, 2013] Nobile, P. N. (2013). *Projeto de um broker de gerenciamento adaptativo de recursos em computação em nuvem baseado em técnicas de controle realimentado [online].* PhD thesis, Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo.

[Oracle, 2013] Oracle (2013). Oracle database: Documentation library. Disponível em: `http://www.oracle.com/pls/db102/homepage`, acessado em 12/09/2013.

[Peixoto, 2012] Peixoto, M. L. M. (2012). *Oferecimento de QoS para computação em nuvens por meio de metaescalonamento.* PhD thesis, Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo.

[Peter Mell and Timothy Grance, 2011] Peter Mell and Timothy Grance (2011). The NIST Definition of Cloud Computing.

[Piatetsky-Shapiro and Connell, 1984] Piatetsky-Shapiro, G. and Connell, C. (1984). Accurate estimation of the number of tuples satisfying a condition. *SIGMOD Rec.*, 14:256–276.

[Pokorny, 2011] Pokorny, J. (2011). Nosql databases: a step to database scalability in web environment. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, pages 278–283, New York, NY, USA. ACM.

[PostgreSQL, 2013] PostgreSQL (2013). Postgresql: The world's most advanced open source database. Disponível em: `http://www.postgresql.org`, acessado em 01/02/2013.

[Pritchett, 2008] Pritchett, D. (2008). Base: An acid alternative. *Queue*, 6(3):48–55.

[QEMU-KVM, 2013] QEMU-KVM (2013). Qemu-kvm: Open source processor emulator. Disponível em: `http://wiki.qemu.org/MainPage`, acessado em 30/01/2012.

[Ramakrishnan and Gehrke, 2008a] Ramakrishnan, R. and Gehrke, J. (2008a). *Sistema de Banco de Dados*, chapter Bancos de Dados Paralelos e Distribuídos, pages 604–636. McGraw-Hill.

[Ramakrishnan and Gehrke, 2008b] Ramakrishnan, R. and Gehrke, J. (2008b). *Sistema de Banco de Dados*, chapter Álgebra e Cálculos Relácionais, pages 83–107. McGraw-Hill.

[Ramakrishnan and Gehrke, 2008c] Ramakrishnan, R. and Gehrke, J. (2008c). *Sistema de Banco de Dados*, chapter Um Otimizador de Consultas Relacionais Típico, pages 399–431. McGraw-Hill.

[Rimal et al., 2009] Rimal, B. P., Choi, E., and Lumb, I. (2009). A taxonomy and survey of cloud computing systems. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, NCM '09, pages 44–51, Washington, DC, USA. IEEE Computer Society.

[Rodrigues, 2013] Rodrigues, P. R. D. (2013). Data warehouses suportados por nuvens. Dissertação de mestrado, Universidade do Minho - Escola de Engenharia.

[Rose, 2004] Rose, R. (2004). Survey of system virtualization techniques. Technical report.

[Schiller et al., 2011] Schiller, O., Schiller, B., Brodt, A., and Mitschang, B. (2011). Native support of multi-tenancy in rdbms for software as a service. In *EDBT*, pages 117–128.

[Shasha and Bonnet, 2002] Shasha, D. and Bonnet, P. (2002). *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science.

[Silberschatz et al., 2010a] Silberschatz, A., Korth, H., and Sudarshan, S. (2010a). *Database Systems Concepts*, chapter Concurrency Control, pages 661–720. McGraw-Hill, Inc., New York, NY, USA, 6 edition.

[Silberschatz et al., 2010b] Silberschatz, A., Korth, H., and Sudarshan, S. (2010b). *Database Systems Concepts*, chapter Query Processing, pages 537–577. McGraw-Hill, Inc., New York, NY, USA, 6 edition.

[Smith, 2010] Smith, G. (2010). *PostgreSQL 9.0 High Performance*, pages 99–124. Packt Publishing, Limited.

[Solaris Zones, 2013] Solaris Zones (2013). Oracle solaris 11 virtualization technology. Disponível em: `http://www.oracle.com/technetwork/server-storage/solaris11/technologies/virtualization-306056.html`, acessado em 12/10/2013.

[Soror et al., 2007] Soror, A. A., Aboulnaga, A., and Salem, K. (2007). Database virtualization: A new frontier for database tuning and physical design. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, ICDEW '07, pages 388–394, Washington, DC, USA. IEEE Computer Society.

[Soror et al., 2008] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S. (2008). Automatic virtual machine configuration for database workloads. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 953–966, New York, NY, USA. ACM.

[Sousa et al., 2011] Sousa, F. R. C., Moreira, L. O., Macêdo, J. A. F. d., and Machado, J. C. (2011). Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios. Technical report, Universidade Federal do Ceará - UFC.

[Stonebraker, 2010] Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4):10.

[Storm et al., 2006] Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., Diao, Y., and Surendra, M. (2006). Adaptive self-tuning memory in db2. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 1081–1092. VLDB Endowment.

[Sullivan et al., 2004] Sullivan, D. G., Seltzer, M. I., and Pfeffer, A. (2004). Using probabilistic reasoning to automate software tuning. *SIGMETRICS Perform. Eval. Rev.*, 32(1):404–405.

[Swami and Gupta, 1988] Swami, A. and Gupta, A. (1988). Optimization of large join queries. In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, SIGMOD '88, pages 8–17, New York, NY, USA. ACM.

[TpcApp, 2012] TpcApp (2012). TCP-H. Disponível em: `http://www.tpc.org/tpch/default.asp`, acessado em 20/03/2012.

[Vaquero et al., 2008] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55.

[Virt Manager, 2014] Virt Manager (2014). VMM: Virtual Machine Manager. Disponível em: `http://virt-manager.org/`, acessado em 15/01/2014.

[Virtual Box, 2013] Virtual Box (2013). Oracle VM VirtualBox. Disponível em: `https://www.virtualbox.org/`, acessado em 15/11/2013.

[VMWARE, 2011] VMWARE (2011). The benefits of virtualization for small and medium businesses. Disponível em: `http://www.vmware.com/files/pdf/VMware-SMB-Survey.pdf`, acessado em 11/10/2013.

[VMWare, 2013] VMWare (2013). Vmware virtualization for desktop and server, application, public and hybrid clouds. Disponível em: `http://www.vmware.com/`, acessado em 15/11/2013.

[VoltDB, 2012] VoltDB (2012). VoltDB: Lightning Fast, Rock Solid. Disponível em: `voltdb.com/`, acessado em 19/01/2012.

[Vouk, 2008] Vouk, M. A. (2008). Cloud computing - issues, research and implementations. *ITI 2008 30th International Conference on Information Technology Interfaces*, 16(4):31–40.

[Wang et al., 2010] Wang, L., Laszewski, G. v., Younge, A., He, X., Kunze, M., Tao, J., and Fu, C. (2010). Cloud computing: a perspective study. *New Generation Computing*, 28:137–146.

[Xen, 2013] Xen (2013). Citrix delivers Cloud Solutions that enable Mobile Workstyles - Citrix. Disponível em: `http://www.citrix.com/`, acessado em 15/11/2013.

[Zhang et al., 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

TARCIZIO ALEXANDRE BINI

ANÁLISE DA APLICABILIDADE DAS REGRAS DE OURO AO TUNING DE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

Tese apresentada como requisito parcial à obtenção do título de Doutor em Ciência da Computação, no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Sfair Sunye

CURITIBA

2014